Blaize.Security

February 10th 2023 / V. 2.0

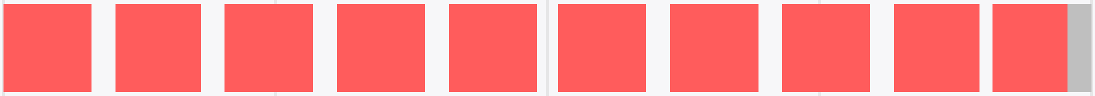
SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

LiquidAccess NFT contract’s source code was taken from the repository provided by the Spectre LiquidAccess NFT team.

SCORE **9.9** /10



The scope of the project for the **1st iteration** included **LiquidAccess NFT** set of contracts**:**

- LiquidAccess.sol:

Repository:

https://github.com/liquidaccess/nft, master branch

Initial commit:

- 87b64635e54488db466424d883d62332ebc4f1ac

Last-audited commit:

- https://github.com/liquidaccess/nft/pull/1, 8ccc6eba28f1fae0a1b2459aa5eead6aa7ae4aca

The scope of the project for the **2nd iteration** included **LiquidAccess NFT** set of contracts**:**

- contracts/LiquidAccess.sol

Repository:

https://github.com/liquidaccess/nft, develop branch

Initial commit:

- 4bcf3bbafb95d0270b189501298133837263bf17

Last-audited commit:

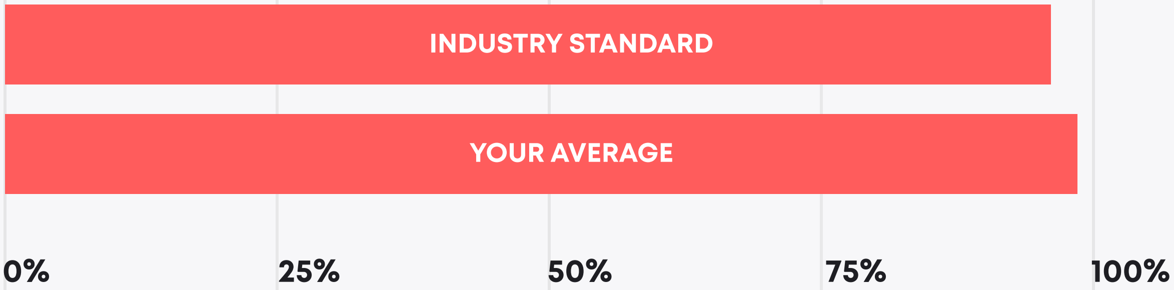
- 539228bcd92b6d049d66e44d13c59d8fb69396a0

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the LiquidAccess NFT protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **LiquidAccess NFT** smart contracts conducted during **November 21st, 2022 - November 25th, 2022.**

2nd iteration of the audit was conducted during **February 2nd, 2023** - **February 10th, 2023.**

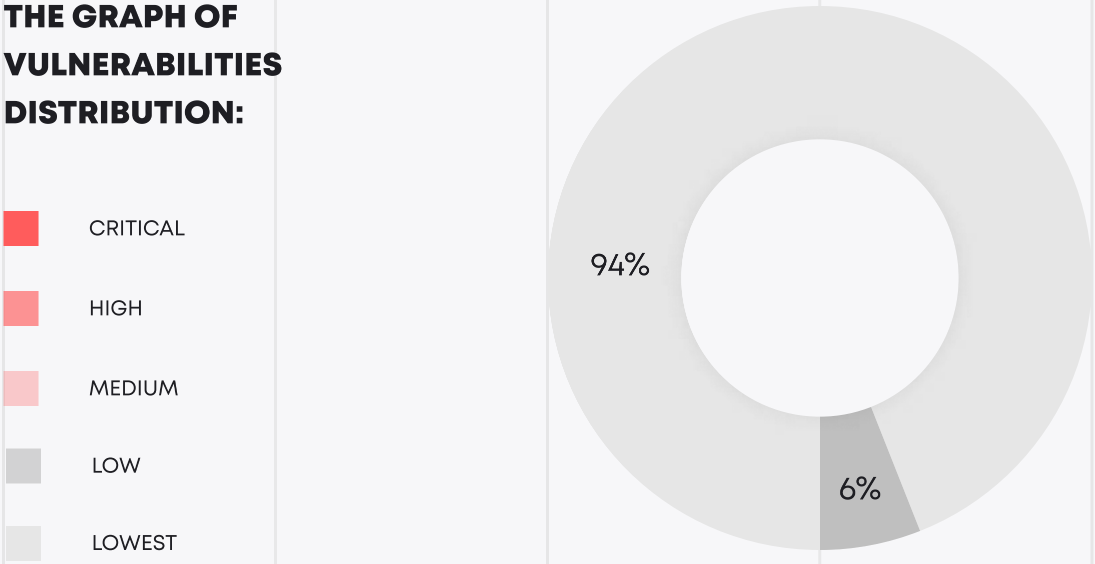
Testable code



The testable code has sufficient coverage, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contracts code, documentation, and requirements presented by the LiquidAccess NFT team. The coverage is calculated based on the set of the Hardhat framework tests and scripts from additional testing strategies. However, in order to ensure full security of the contract, the Blaize.Security team suggests the LiquidAccess NFT team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 14 problems were found. No critical issues were found. Most of the issues were fixed by the LiquidAccess NFT team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	1	1
Lowest	14	13

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, but best practices should be implemented.

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contract for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contract by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with these tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

During the audit, the Blaize security team carefully checked the core contract of LiquidAccess NFT - LiquidAccess.sol. The goal of the audit was to verify that the best Solidity practices are applied, the contract corresponds to the ERC721 standard. It was also required to ensure the safety of minting, transfering, and other standard processes of an NFT contract.

No critical issues were found in the contract. Though, one low and several informational problems were discovered. The low issue was connected to the absence of one validation parameter, during which tokens can't be transferred. The informational issues were connected to the violation of the Solidity code style, optimizations, and bussiness-logic features validations. Most of the issues were succesfully fixed by the LiquidAccess NFT team.

The overall security of the contract is high enough. The contract corresponds to the NFT standard in a secure way. The provided repository contains a sufficient tests coverage provided by the LiquidAccess NFT team. The Blaize security team has also prepared their own set of additional testing scenarios.

During the second iteration audit, the Blaize security team checked changes in the LiquidAccessNFT contract. The team added new functionality and interfaces; therefore, NFT received new methods. During the audit (after the notice from the auditors), the LiquidAccess team added a public burn() function. In this way, they have closed several info issues from the previous iteration. Despite functioning burn(), auditors should notice that it creates a possible vulnerability for NFTs burn from the upgradeable contracts. Though, the risk of the exploit is low, and auditors advise providing sanitizing policy for the protocols and marketplaces that will utilize these NFTs.

During the 2nd iteration, auditors found several info issues and prioritized unresolved issues from the previous iteration again. The LiquidAccess team resolved all mentioned issues except the one connected with the global lock time update.

Auditors should also mention that one of the most important changes in the NFT is functionality for the NFT URI update, which is delegated to the minter role.

The overall security of the contract is still high enough, and the contract corresponds to the NFT standard securely.

	RATING
Security	9.9
Gas usage and logic optimization	10
Code quality	9.8
Test coverage*	9.9
Total	9.9

* LiquidAccess NFT team has prepared a solid set of unit-tests. Blaize security team has prepared their own set of additional test-cases as well.

CONTRACT OVERVIEW

LiquidAccess.sol is an NFT contract that implements the ERC721 NFT standard, ERC2981 royalty standard, ERC721Burnable, ERC721URIStorage, and ERC4906 Metadata Update Extension. During the contract deployment, the token's name, symbol, merchant, and merchant ID are set in the storage.

The minting flow of the contract contains the safeMint() function and batchMint(), which can be executed only by the minter of the contract. During the safeMint() function, the minter can specify the recipient, subscription type, and token expiration. During batchMint() function, the minter can mint a series of NFTs and send them to the recipient accounts. Also, the contract contains the burn function now (added during the latest update). There are also setters, which allow the owner to change the subscription type and expiration of the existing tokens. Also, the contract contains additional setters, enabling the owner to set the following information about the contract: royalty, lockup period, users and NFTs blacklist, NFT and contract's name, description, and image.

There is also a blacklist for users and NFTs in the contract. The owner can add or remove users and NFTs from the blacklist. Blacklisted users are forbidden from transferring or receiving NFTs, while blacklisted tokens can't be transferred.

Additionally, there is a lockup period between transfers of the tokens. For example, if the lockup period is set to 1 day, each NFT can be transferred only once a day.

COMPLETE ANALYSIS (1ST ITERATION)

LOW-1	✓ Resolved

Lock period can be set to large values.

LiquidAccess.sol: setLockupPeriod().
There is no validation in this function that the `period` parameter is not equal to large values. In case a large value is passed in this function, it can potentially block any other transfers of the NFT (since the lock up period of the NFT isn't updated when global _lockupPeriod`is updated). The issue is marked as low since the owner should validate which value is passed in this function. Though it is still recommended to add a maximum limit which the `period` parameter could not exceed.

Recommendation:

Validate that the `period` parameter doesn't exceed a certain value.

LOWEST-1	✓ Resolved

Blacklist mappings can use boolean values to indicate if the value is blacklisted.

LiquidAccess.sol: lines 22, 23
Mapping `mapping(address => address)`can be changed to `mapping(address =>bool)`, and `mapping(uint256 => uint256)`to `mapping(uint256 => bool)`. Though the current solution works well and doesn't lead to any vulnerabilities, using booleans instead will improve readability and code clarity.

Recommendation:

Change mappings and relevant functions.

LOWEST-2	✓ Resolved

Some variables are changed without an event.

LiquidAccess.sol: functions setLockupPeriod(), addNFTToBlacklist(), removeNFTFromBlacklist(), addAddressToBlacklist(), removeAddressFromBlacklist(), setContractName(), setContractDescription(), setContractImage().
Mapping and variables inside the LiquidAccess contract can be changed without an event. Thus, it can be complicated to parse and update data related to this contract.

Recommendation:

Add events for every storage change. It is advised to indicate the previous and new values.

LOWEST-3	✓ Resolved

Explicit getters can be omitted in favor of the default ones.

LiquidAccess.sol: lines 16-23, 121-122, 247-249
The variables can be set to public, so default getters will be created. Since this contract isn't inherited by any other contracts, there is no point of making these variables private. Thus, security is not a concern in this case.

Recommendation:

Remove explicit getters and make the variables public.

Post-audit:

Issue was resolved during the 2nd audit iteration

LOWEST-4 ✓ Resolved

Style guide violation.

LiquidAccess.sol: 121-122, 247-249, etc
Solidity style guide (the order of layout) is violated, which makes the code harder to read.

Recommendation:

Change your contract so as to comply with the style guide (especially the order of layout). You can also split the contract into several parts to divide logic and variables for readability.

LOWEST-5 ✓ Resolved

Functions can be marked as external.

LiquidAccess.sol: functions contractURI(), lockupLeftOf(), lockupPeriod(), isNFTBlacklisted(), isAddressBlacklisted(), merchantName(), merchantId(). In order to decrease gas spending, some of public functions that aren't used within other functions can be marked as external.

Recommendation:

Mark the aforementioned functions as external.

LOWEST-6 ✓ Resolved

Conditions can be united in one `if.`

LiquidAccess.sol: function lockupLeftOf(), lines 126,129.
Since `0`value is returned in both branches, they can be united to improve code readability.

Recommendation:

Unite conditions of `if`s with || operator.

LOWEST-7	Unresolved

Personal NFT lock up is not updated in case global lock up period is updated. LiquidAccess.sol

When `_lockupPeriod`is updated, the personal lockup of each NFT that is currently locked for transfers is not updated. The issue is marked as info since it doesn't expose any danger and might be a part of the bussiness logic.
It impacts the logic in _beforeTokenTransfer function where the check is performed before the transfer and NFT may stay locked even if locktime is set to 0. It also may impact the usage of lockupLeftOf() function It can be resolved by additional check against the _lockupPeriod being set.

Recommendation: Verify that the personal lockup for each NFT should not be affected when the global lockup period is changed. OR add additional check in _beforeTokenTransfer () and lockupLeftOf() against the _lockupPeriod.

LOWEST-8	✓ Verified

Users and NFTs can be blacklisted.

Though such functionality is a part of the bussiness logic and isn't considered as security threat, it should be noted that the owner of the contract can blacklist any user and NFT so that such a user can't transfer or receive NFTs and such an NFT can't be transferred.
Post-audit: Verified to be a part of the bussiness logic in order to prevent any suspicous actions on the contract.

LOWEST-9	✓ Resolved

The owner can blacklist a non-existing NFT.

LiquidAccess.sol: function addNFTToBlacklist().
Though only the owner can execute this function and validate that an existing NFT is passed only, it is recommended to validate that a provided `_nft`exists.

Recommendation:

Validate that the `_nft` parameter exists before blacklisting.

LOWEST-10	✓ Resolved

Tokens cannot be burned.

LiquidAccess inherits ERC721 contract, which contains internal _burn() function but it does not implement public burn function. LiquidAccess overrides default _burn() and has checks for burn in beforeTransfer function, but does not implement the public function for NFTs burn.
Note: Issue was updated during the 2nd audit iteration.

Recommendation:

Verify that you do not need burn() functionality and so unnecessary code can be removed OR implement public burn functionality if you need such.

Post-audit:

"Burn" was added with no restrictions. Therefore, the owner of the NFT can freely burn it. Auditors should note that unrestricted burn can open an exploit for upgradeable contracts that will hold NFT. So, it is additionally recommended to provide sanitizing policy for 3rd party contracts which may hold NFTs; or limit NFTs usage only within the protocol and trustable marketplaces.

LOWEST-11	✓ Resolved

Unreachable branch at line 103.

There is no need to check transfer to zero address because error "ERC721: transfer to the zero address" will be raised at ECR721 contract.
Note: issue was resolved during the 2nd audit iteration (with added burn()
Recommendation: Remove the unreachable branch.

COMPLETE ANALYSIS (2ND ITERATION)

LOWEST-1 ✓ Resolved

Custom errors should be used.

LiquidAccess.sol: permit(), lines 263-267; setLockupPeriod(), line 355; _beforeTokenTransfer(), lines 445, 450, 455, 458.
Starting from the 0.8.4 version of Solidity, it is recommended to use custom errors instead of “require” statements and storing error message strings in storage. Custom errors are more efficient in terms of gas spending and they increase code readability.

Recommendation:

Use custom errors.

LOWEST-2 ✓ Resolved

Variables should be immutable.

LiquidAccess.sol: variables _merchantName, _merchantId.
It is recommended to have variables marked as public and immutable - in case if they are set just once in the constructor .
Also, additional getters are unnecessary, since more efficient public getters are generated for public variables by default.

Recommendation:
Mark variables as public (and remove extra getters) and immutable. Or verify that existent visibility is necessary and only immutability is required

LOWEST-3	✓ Resolved

Variable is not visible.

LiquidAccess.sol: _tranferFromCounter variable.
The variable is marked as private, despite the fact that it can be updated. But the user can access variable value only through the emitted event or via checking the info in the explorer. Such an approach may be inconvenient for the frontend or user experience.

Recommendation:
Verify that the variable should stay private and accessible only through events OR make it public.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE BLAIZE.SECURITY TEAM (1ST ITERATION)

- ✓ supports interfaces (60ms)

NFT transfer

- ✓ user can transfer their NFT (77ms)
- ✓ NFT cannot be transfered to zero address (55ms)
- ✓ user cannot transfer their blacklisted NFT (74ms)
- ✓ user cannot transfer their blacklisted NFT by approve (85ms)
- ✓ NFT from blacklisted address cannot be sent by approve (65ms)
- ✓ user cannot transfer their NFT to blacklisted address (46ms)
- ✓ user cannot transfer their NFT if they are blacklisted (44ms)

Setters

- ✓ sets contract image (77ms)

setExpirationDate

- ✓ only the owner can set an expiration date (46ms)
- ✓ expiration date can be set only for existing token

setSubscriptionType

- ✓ only the owner can set subscription type (46ms)
- ✓ subscription type can be set only for existing token

setContractImage

- ✓ only the owner can set image

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
LiquidAccess.sol	100	98.53	100	100

```
Contract: LiquidAccess
    .tokenExists() modifier
    ✓ Should correct passing (60ms)
    ✓ Should correct reverting at .tokenURI()
    ✓ Should correct reverting at .lockupLeftOf()
    ✓ Should correct reverting at .isNFTBlacklisted()
    ✓ Should correct reverting at .addNFTToBlacklist()
    ✓ Should correct reverting at .removeNFTFromBlacklist()
    .batchMint() function
    ✓ Should correct minting (55ms)
    ✓ Should revert if sender hasn`t MINTER_ROLE (59ms)
    ✓ Should revert if recipients length and uris length not equal
    ✓ Should not minting for users who is in Blacklist (42ms)
    .changeTokenUri() function
    ✓ Should changing (67ms)
    ✓ Should revert if sender hasn`t MINTER_ROLE (85ms)
    ✓ Should revert if token`s ID not Exist (68ms)
    .updateAllTokensMetadata() function
    ✓ Should update (56ms)
    ✓ Should revert if sender hasn`t MINTER_ROLE (76ms)
    ✓ Should do nothing is contract hasn`t token
    .setContractImage() function
    ✓ Should revert if sender hasn`t MINTER_ROLE (85ms)
    .supportsInterface() function
    ✓ Should correct work if interface code is '0x49064906'
    .permit() function
    ✓ Should revert if approve to owner (41ms)
    ✓ Should revert if 'v' value incorrect (not equal 27 or 28) (39ms)
    ✓ Should revert if 'signer' value incorrect (40ms)

    21 passing (3s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
LiquidAccess.sol	100	100	100	100

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE LIQUIDACCESS NFT

```
Contract: LiquidAccess
    Contract info
    ✓ should have the correct name (39ms)
    ✓ should have the correct symbol
    Merchant info
    ✓ should return merchant name
    ✓ should return merchant id
    Token minting
    ✓ should safeMint (39ms)
    ✓ shoud return correct tokenId (176ms)
    ✓ should emit Transfer event (42ms)
    ✓ should revert if not owner (58ms)
    Token info
    ✓ should have the correct subscription type (49ms)
    ✓ should be able to change subscription type
    ✓ should have the correct expiration date
    ✓ should be able to change expiration date
    ✓ should revert if token does not exist
    Transfer
    ✓ should emit TransferFrom event with transfer counter (79ms)
    ✓ should revert if not token owner (60ms)
    SafeTransfer
    ✓ should emit TransferFrom event with transfer counter (86ms)
    ✓ should revert if not token owner
    Approved transfer
    ✓ should be able to approve an address for a transfer
    ✓ should be able to transfer by approved address (93ms)
    Transfer Lockup
    ✓ should be able to set lockup period
    ✓ should revert if not owner
    ✓ should lock transfers after each transfer (64ms)
    ✓ should be able to retrieve lockup period of a token (63ms)
    ✓ should unlock transfers after lockup period (77ms)
    ✓ should not revert if lockup period is 0 (59ms)
    Royalty
    ✓ should return 5% royalty by default
    ✓ should be able to change royalty recipient
    ✓ should be able to change royalty fee
    ✓ should be able to remove royalty
    ✓ should revert if caller is not owner
    NFT blacklisting
    ✓ should be able to blacklist NFT (100ms)
    ✓ should be able to remove NFT from blacklist (110ms)
    ✓ should revert if caller is not owner
    ✓ should not be able to transfer blacklisted NFT
    Address blacklisting
    ✓ should be able to blacklist address (189ms)
    ✓ should be able to remove address from blacklist (113ms)
    ✓ should revert if caller is not owner
    ✓ should not be able to transfer NFT to blacklisted address
    ✓ should not be able to transfer NFT from blacklisted address
    User tokens
    ✓ should be able to retrieve user tokens (114ms)
    Metadata
    ✓ should be able to change NFT meta name (64ms)
    ✓ should be able to change NFT meta description (65ms)
    ✓ should be able to change NFT meta image (48ms)
    ✓ should have correct NFT meta attributes (47ms)
    ✓ should revert if caller is not owner
    Contract metadata
    ✓ should be able to change contract meta name (45ms)
    ✓ should be able to change contract meta description (40ms)
    ✓ should use nft image as contract image
    ✓ should contain royalty info (40ms)
    ✓ should revert if caller is not owner
```

```
Interface support
  ✓ should support ERC165
  ✓ should support ERC721
  ✓ should support ERC721Metadata
  ✓ should support ERC721Enumerable
  ✓ should support ERC2981


  55 passing (8s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
LiquidAccess.sol	100	88,24	97,06	98,65

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE BLAIZE.SECURITY TEAM (2ND ITERATION)

Contract: LiquidAccess

Token Burn

- ✓ should be able to burn existing token (63ms)
- ✓ fails burning non existing token
- ✓ Allows for the side contract to burn a token (100ms)

Batch minting

- ✓ should deliver NFTs to recipients (70ms)
- ✓ should continue enumeration (132ms)
- ✓ should not allow owner to mint (115ms)
- ✓ should not mint to blacklisted users (65ms)
- ✓ no error when minting to non ERC721Receiver contracts (unfortunately) (52ms)

ERC2612: Permit

- ✓ should not allow to transfer to marketplace without permission
- ✓ should check the permission
- ✓ should check the nonce, not allowing to re-use same signature (56ms)
- ✓ should check the deadline
- ✓ when signature is OK, permission works (48ms)

NFT blacklisting

- ✓ should be able to blacklist NFT (98ms)
- ✓ should be able to remove NFT from blacklist (108ms)
- ✓ should revert if caller is not owner (82ms)
- ✓ should not be able to transfer blacklisted NFT (43ms)

Address blacklisting

- ✓ should be able to blacklist address (87ms)
- ✓ should be able to remove address from blacklist (100ms)
- ✓ should revert if caller is not owner (78ms)
- ✓ should not be able to transfer NFT to blacklisted address (46ms)
- ✓ should not be able to transfer NFT from blacklisted address

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
LiquidAccess.sol	96,97	80,49	96,43

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol, or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool that helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.