Aave
Permissioned market
SMART CONTRACT AUDIT

September 17, 2021

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Aave. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 SECURITY ASSESSMENT METHODOLOGY

A group of auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 Project architecture review:
> Reviewing project documentation
> General code review
> Reverse research and study of the architecture of the code based on the source code only
> Mockup prototyping

Stage goal:
Building an independent view of the project's architecture and identifying logical flaws in the code.

02 Checking the code against the checklist of known vulnerabilities:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code
> Checking with static analyzers (i.e Slither, Mythril, etc.)

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the code for compliance with the desired security model:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit
> Exploits PoC development using Brownie

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of interim auditor reports into a general one:
> Cross-check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level and provide the client with an interim report.

05 Bug fixing & re-check:
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.3 EXECUTIVE SUMMARY

Aave is a decentralized non-custodial liquidity markets protocol where users can participate as depositors or borrowers. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an overcollateralized (perpetually) or undercollateralized (one-block liquidity) fashion.Audited smart contracts are designed to work with contract access rights. In a dedicated PermissionManager contract, there is logic for setting, removing, and checking roles for addresses. Other contracts provide access rights to use the functionality.Here is a brief description of the functional in the contracts under study:

- `PermissionManager` - This smart contract implements basic whitelisting functions for different actors of the permissioned protocol.
- `PermissionedLendingPool` - This smart contract adds a permission layer to the LendingPool contract to enable whitelisting of the users interacting with it.
- `PermissionedStableDebtToken` - This smart contract implements a stable debt token to track the borrowing positions of users at stable rate mode, with permissioned roles on credit delegation.
- `PermissionedVariableDebtToken` - This smart contract implements a variable debt token to track the borrowing positions of users at variable rate mode, with permissioned roles on credit delegation.
- `WETHGateway` - This smart contract allows you to make and remove a deposit in WETH (deposits WETH into the reserve, using native ETH. A corresponding amount of the overlying asset (aTokens) is minted).
- `PermissionedWETHGateway` - This smart contract is inherited from `WETHGateway`. In this contract, the developers have added functionality for role-activated access to the main functions of the contract.
- `LendingPoolCollateralManager` - This smart contract implements actions involving management of collateral in the protocol, the main one being the liquidations.

1.4 PROJECT DASHBOARD

Client	Aave
Audit name	Permissioned market
Initial version	7ebd95e22e4c901becfd2515f366167891ae81c8
Final version	303600a5260c11e1ca7027b9f0bfdcae99ec406b
Date	May 21, 2021 - September 17, 2021
Auditors engaged	3 auditors

FILES LISTING

LendingPoolCollateralManager.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/protocol/lendingpool/LendingPoolCollateralManager.sol
PermissionedLendingPool.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/protocol/lendingpool/PermissionedLendingPool.sol
PermissionManager.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/protocol/configuration/PermissionManager.sol
PermissionedStableDebtToken.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/protocol/tokenization/PermissionedStableDebtToken.sol
PermissionedVariableDebtToken.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/protocol/tokenization/PermissionedVariableDebtToken.sol
PermissionedWETHGateway.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/misc/PermissionedWETHGateway.sol
WETHGateway.sol	https://github.com/aave/protocol-v2/blob/7ebd95e22e4c901becfd2515f366167891ae81c8/contracts/misc/WETHGateway.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	2
Warning	9
Comment	7

CONCLUSION

During the audit no critical issues were found, several majors, warnings and comments were spotted. After working on the reported findings all of them were fixed by the client or acknowledged.Final commit identifier with all fixes: `303600a5260c11e1ca7027b9f0bfdcae99ec406b`

2. FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

MJR-1	Incorrect logic for access modifier
File	PermissionedLendingPool.sol PermissionManager.sol
Severity	Major
Status	Fixed at 7a2eece3

DESCRIPTION

At line PermissionedLendingPool.sol#L256 the `onlyUserPermissionAdmin` access modifier is used for the `seize()` function. It checks that the sender of the `msg.sender` transaction is a permission administrator for user `user`. Look at line PermissionManager.sol#L176.

Let's assume the owner has removed this permission administrator. This can be done with the `removePermissionAdmins()` method on these lines: PermissionManager.sol#L38-L44. The data is modified only for the `_permissionsAdmins` variable while for variable `_users[user].permissionAdmin`, it does not change.

For the `_isPermissionAdminOf()` function, you need to do an additional check by calling the `isUserPermissionAdminValid` function located on lines: PermissionManager.sol#L180-L182.

RECOMMENDATION

It is recommended to correct this error.
You will also need to implement the logic for the correct removal of a permission administrator from all users. But this relationship will need to be stored in a separate variable. For example, this should be a variable of type:

```
mapping (address => address[]) private _usersWithPermissionAdmin;
```

But do not forget that such users will always have an associated permission administrator.

CLIENT'S COMMENTARY

> Condition for `_isPermissionAdminOf()` was changed in 7a2eece3. There will be no additional changes made. It may be impossible to change the permissionAdmin field for each user onchain, depending on the number of users, and also it may be useful to keep track of the address of the permission admin that added a certain user.

MJR-2	Incorrect values in events
File	PermissionedStableDebtToken.sol PermissionedVariableDebtToken.sol
Severity	Major
Status	Acknowledged

DESCRIPTION

When minting new tokens and when burning existing tokens, you need to record the events of `Transfer`.
At line PermissionedStableDebtToken.sol#L239, function `_mint()` is called.
At line PermissionedStableDebtToken.sol#L252, function `_burn()` is called.
In functions `_mint ()` at line
PermissionedStableDebtToken.sol#L404
and `_burn()` at line
PermissionedStableDebtToken.sol#L423
there is no recording of events `Transfer`.
Thus, this event `Transfer` is recorded only once on line
PermissionedStableDebtToken.sol#L256.
The value of the variable `amount` is not equal to the values of the variable `balanceIncrease.sub(amount)` and `amount.sub(balanceIncrease)`.

Similarly in another contract:

- at line PermissionedVariableDebtToken.sol#L111 the `Transfer` event is written, `amount` is passed, and `amount.rayDiv(index)` is passed to the `_mint()` function.
- at line PermissionedVariableDebtToken.sol#L13 the `Transfer` event is written, `amount` is passed, and `amount.rayDiv(index)` is passed to the `_burn()` function.

RECOMMENDATION

It is recommended to fix the errors found.

CLIENT'S COMMENTARY

> Acknowledged. This is a known issue caused by the accrual of interest of the aTokens/debtTokens and changing it would be out of the scope of the project.

2.3 WARNING

WRN-1	No logic for returned values from the function
File	LendingPoolCollateralManager.sol PermissionedLendingPool.sol
Severity	Warning
Status	Fixed at 303600a5

DESCRIPTION

At lines LendingPoolCollateralManager.sol#L75-L91 there is a `seize()` function. It returns two values of type `uint256` and `string`.
But there is no initialization of these variables anywhere.
For line PermissionedLendingPool.sol#L262, this function is called. And on lines 267-269, the return values are processed.
But with the current logic for the `seize()` function, the condition on line PermissionedLendingPool.sol#L269 will be satisfied in all cases.

RECOMMENDATION

It is recommended to correct this error.

WRN-2	Move event emit to another location
File	PermissionedStableDebtToken.sol PermissionedVariableDebtToken.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

At line
PermissionedStableDebtToken.sol#L174
the `_mint ()` function is called and then the `Transfer` event is recorded.
It is recommended to move the `Transfer` event record to the `_mint()` function on line
PermissionedStableDebtToken.sol#L404
for the following reasons::

1. The `_mint ()` function and the `Transfer` event must be called together;
2. If you record the `Transfer` event after calling the `_mint()` function, you can forget to do it;
3. If you record the `Transfer` event after calling the `_mint()` function, then you can transfer incorrect values there;
4. The `Transfer` event is called from the`_mint ()`function in the well-known Openzeppelin library.

Similar code was found in the following locations.
At line PermissionedStableDebtToken.sol#L239 there is a call to the `_mint()` function and only then on line 256 a record of the `Transfer` event is made.
At line PermissionedStableDebtToken.sol#L252 there is a call to the `_burn()` function and only then on line 256 a record of the `Transfer` event is made.

At line PermissionedVariableDebtToken.sol#L109 there is a call to the `_mint()` function and only then on line 111 a record of the `Transfer` event is made.
At line PermissionedVariableDebtToken.sol#L132 there is a call to the `_burn()` function and only then on line 134 a record of the `Transfer` event is made.

RECOMMENDATION

It is recommended to transfer events to another location.

CLIENT'S COMMENTARY

> Acknowledged. No changes will be made as it requires changes to the core Aave protocol.

WRN-3	Functions similar in functionality have different logic for return values
File	PermissionedStableDebtToken.sol PermissionedVariableDebtToken.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

The `mint()` function is used to create new tokens. The opposite function `burn()` is used to burn existing tokens.
These functions do the opposite, and both are designed to work with tokens.

However, in the `mint()` function on line PermissionedStableDebtToken.sol#L141 there is a return variable of type `bool`
and there are no return variables in the `burn ()` function on line PermissionedStableDebtToken.sol#L197.

Likewise, for the `mint()` function on line PermissionedVariableDebtToken.sol#L100 and for the `burn()` function on line PermissionedVariableDebtToken.sol#L128.

RECOMMENDATION

It is recommended to do the same logic for the returned variables.

CLIENT'S COMMENTARY

> Acknowledged. From our perspective mint/burn functions are used internally within the protocol and aren't exposed to developers, and not being part of the ERC20 standard there is no need to respect any specific interface. We will make sure to add more details to the natspec comments to explain the rationale behind the return logic.

WRN-4	Invalid variable return value from the function
File	PermissionedStableDebtToken.sol PermissionedVariableDebtToken.sol LendingPool.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

At line PermissionedStableDebtToken.sol#L189, the value returned by the `mint()` function is processed.
The first time this function is called, it will return `true`. But on subsequent calls to this function, if the balance of the `onBehalfOf` wallet has a positive value of tokens, the value`false` will be returned.
It is not right.

Similarly, on line PermissionedVariableDebtToken.sol#L114, the value returned by the `mint()` function is processed.
The first time this function is called, it will return `true`. But on subsequent calls to this function, if the balance of the `onBehalfOf` wallet has a positive value of tokens, the value `false` will be returned.

Line LendingPool.sol#L121 has similar handling of the return value. But then it is necessary to give clear names for the variables.
This can lead to the fact that another developer will not use the received answer to his logic correctly.

RECOMMENDATION

It is recommended to correct this error.

CLIENT'S COMMENTARY

> Acknowledged. From our perspective mint/burn functions are used internally within the protocol and aren't exposed to developers, and not being part of the ERC20 standard there is no need to respect any specific interface. We will make sure to add more details to the natspec comments to explain the rationale behind the return logic.

WRN-5	No validation of the address parameter value in contract constructor
File	WETHGateway.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

The variable is assigned the value of the constructor input parameter. But this parameter is not checked before this. If the value turns out to be zero, then it will be necessary to redeploy the contract, since there is no other functionality to set this variable.

- At line WETHGateway.sol#L27 the `WETH` variable is set to the value of the `weth` input parameter.

RECOMMENDATION

It is necessary to add a check of the input parameter to zero before initializing the variables.

CLIENT'S COMMENTARY

> Acknowledged, no action. Passing zero address is an edge case as much as assigning a wrong address so checking for nonzero address only solves a very particular edge case.

WRN-6	It is necessary to compare the values of the variables calculated in different places
File	WETHGateway.sol LendingPool.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

At line WETHGateway.sol#L104
the `repay()` function is called near the address `lendingpool`.
At lines:

- LendingPool.sol#L204
- LendingPool.sol#L946
  it shows that the function returns a variable of type `uint256`.
  But this value is not processed.
  The `paybackAmount` variables are calculated twice in different places. It is recommended to compare the values of these variables.

RECOMMENDATION

Add processing of the value returned by the function.

CLIENT'S COMMENTARY

> Acknowledged. This contract is not part of the core protocol so no actions will be performed.

WRN-7	The value returned by the token transfer function is not processed
File	WETHGateway.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

According to the ERC-20 standard, functions for working with tokens return a variable of type `bool`.
But on these lines there is no processing of the received values:

- WETHGateway.sol#L31
- WETHGateway.sol#L69
- WETHGateway.sol#L156

RECOMMENDATION

Add processing of the value returned by the function.

CLIENT'S COMMENTARY

> Acknowledged. This contract is not part of the core protocol so no actions will be performed.

WRN-8	Possible `withdrawETH` is missing in `PermissionedLendingPool`
File	WETHGateway.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

`msg.sender` and `to` of the WETHGateway.sol#L56 function will not be checked for access.

RECOMMENDATION

It is recommended to add the `withdrawETH` function in the `PermissionedLendingPool`.

```
function withdrawETH(
    address lendingPool,
    uint256 amount,
    address to
) public payable virtual override {
    ILendingPool pool = ILendingPool(lendingPool);

    require(_isInRole(msg.sender, DataTypes.Roles.DEPOSITOR, pool), Errors.USER_UNAUTHORIZED
    require(_isInRole(to, DataTypes.Roles.DEPOSITOR, pool), Errors.USER_UNAUTHORIZED);

    super.withdrawETH(lendingPool, amount, to);
}
```

CLIENT'S COMMENTARY

> The permissioned lending pool does not deal with ETH directly so it does not need any `withdrawETH()` function. The `withdrawETH()` of the `WETHGateway` is only used as a UX simplification for users coming with ETH to use the protocol.

WRN-9	Invalid values for variable `_borrowAllowances`
File	PermissionedStableDebtToken.sol PermissionedVariableDebtToken.sol
Severity	Warning
Status	No Issue

DESCRIPTION

At line PermissionedStableDebtToken.sol#L145 the function call is made `_decreaseBorrowAllowance()` to change the value of the variable `_borrowAllowances`.
Here the value of the variable `_borrowallowances` changes, which is dependent on the value of the variable `amount`.

However, on line PermissionedStableDebtToken.sol#L174 there is a mint of new tokens in the amount of `amount.add(BalanceIncrease)`.
Thus, the value of `balanceIncrease`.

Likewise, at line PermissionedVariableDebtToken.sol#L102 the function call is made `_decreaseBorrowAllowance()` to change the value of the variable `_borrowAllowances`.
Here the value of the variable `_borrowallowances` changes, which is dependent on the value of the variable `amount`.

However, on line PermissionedVariableDebtToken.sol#L109 there is a mint of new tokens in the amount of `amount.rayDiv(index)`.

This will lead to errors in the program.

RECOMMENDATION

It is recommended to fix the errors found.

CLIENT'S COMMENTARY

> `borrowAllowance` is decreased for the delegatee (user) while debt tokens are minted for accounting to the delegator (onBehalfOf) by design. We can't identify any issue here.

2.4 COMMENT

CMT-1	Function is too long
File	LendingPoolCollateralManager.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At lines LendingPoolCollateralManager.sol#L104-L268 the `liquidationCall()` function is contained.
According to the SOLID principles (https://en.wikipedia.org/wiki/SOLID) made for successful programming, each function should be responsible for only one action.
It will be possible to split the logic of the `liquidationCall()` function into separate functions and from the `liquidationCall()` function call these modules.
Each subfunction should only perform one operation. It must do it well and it she shouldn't do anything else.

RECOMMENDATION

It is recommended to split the function into sub-functions that perform only one action.

CLIENT'S COMMENTARY

> Acknowledged. There have been several attempts to refactore the `liquidationCall()` function but it's not so easy given the nature of the action and all the different conditions that might happen and must be checked.

CMT-2	Too many input parameters for functions
File	LendingPoolCollateralManager.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At lines LendingPoolCollateralManager.sol#L295-L302 there is a description of the input parameters for the `_calculateAvailableCollateralToLiquidate()`. There are 6 variables here.
But in programming functions with three arguments (ternary) should be avoided whenever possible. The need for functions with a large number of arguments (polyary) should be supported by very good reasons - and still it is better not to use such functions.
This is all covered in the [`Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin`].(https://www.oreilly.com/library/view/clean-code-a/9780136083238/)

RECOMMENDATION

It is recommended to reduce the number of input parameters at least for internal functions.

CLIENT'S COMMENTARY

> Acknowledged. Those parameters are needed for the calculations so we believe it is acceptable in this case.

CMT-3	Using hardcoded constants instead of constants from Errors library
File	PermissionManager.sol Errors.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At line PermissionManager.sol#L66 and at line PermissionManager.sol#L99
constant `INVALID_PERMISSIONADMIN` already exists in Errors.sol#L111 and can be used.

RECOMMENDATION

It is recommended to use constans from libraries. Also `INVALID_ROLE` constant can be moved into Errors library.

CLIENT'S COMMENTARY

> For the permission manager, it is meant as a standalone contract that can be eventually reused so it was preferred to avoid tighting it to the Errors library that is protocol specific. We could not find `INVALID_ROLE` in the errors library.

CMT-4	Inappropriate error used
File	PermissionedLendingPool.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

The `LP_LIQUIDATION_CALL_FAILED` error constant for `liquidationCall()` function is used in `seize()` funciton at line PermissionedLendingPool.sol#L265.

RECOMMENDATION

It is recommended to create the `PLP_SEIZE_FAILED` error constant for `seize()` function and use it.

CLIENT'S COMMENTARY

> Acknowledged. The `PermissionedLendingPool` contract size is very close to the size limit of 24KB (24522B) and something weird happens on compilation. When replacing the `LP_LIQUIDATION_CALL_FAILED` with `PLP_SEIZE_FAILED`, the contract size increases by 100 bytes which is far more than what you would expect by just adding a new simple constant. We tried to reduce the code size enough to allow this to be fixed with some small refactorings but without success.

CMT-5	Debt tokens code duplication
File	PermissionedStableDebtToken.sol StableDebtToken.sol PermissionedVariableDebtToken.sol VariableDebtToken.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

Pairs PermissionedStableDebtToken.sol#L18-StableDebtToken.sol#L18 and PermissionedVariableDebtToken.sol#L17-VariableDebtToken.sol#L17 have the same code except base `PermissionedDebtTokenBase` or `DebtTokenBase`.
The problem is that all code modifications have to be implemented in both permissioned or original versions.

RECOMMENDATION

It is recommended to add `VariableDebtTokenBase` and `StableDebtTokenBase` contracts and inherit from them.

CLIENT'S COMMENTARY

> Acknowledged. This was a choice, because unfortunately the structure of the inheritance doesn't make it easy to inherit `PermissionedStableDebtToken` and `PermissionedVariableDebtToken` directly from `VariableDebtTokenBase` and `StableDebtTokenBase`. A bigger refactoring that also involves these base classes and the original `StableDebtToken` and `VariableDebtToken` is needed, which was not considered convenient for this release.

CMT-6	Possible code improving
File	PermissionedWETHGateway.sol PermissionedDebtTokenBase.sol PermissionedLendingPool.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

Using `onlyDepositors` and `onlyBorrowers` and `PERMISSION_MANAGER` constant in PermissionedWETHGateway.sol#L40-L80 as in PermissionedDebtTokenBase.sol#L23-L35 or PermissionedLendingPool.sol#L17-L48 will make the code more consistent.

RECOMMENDATION

It is recommended to add modifiers `onlyDepositors` and `onlyBorrowers` and store `keccak256('PERMISSION_MANAGER')` as constant.

CLIENT'S COMMENTARY

> Acknowledged. The `WETHGateway` is a simple helper contract and we believe the current implementation is clear enough.

CMT-7	Missing functions documentation
File	PermissionManager.sol IPermissionManager.sol WETHGateway.sol IWETHGateway.sol PermissionedStableDebtToken.sol IStableDebtToken.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

For the PermissionManager.sol contract, the description of functions and events is located in the IPermissionManager.sol.

But for the WETHGateway.sol contract, the description of functions and events is not in the IWETHGateway.sol interface. It's in the contract itself.

Also for the PermissionedStableDebtToken.sol contract, the description of functions and events is located in the IStableDebtToken.sol interface and the contract itself.

RECOMMENDATION

It is recommended to use same code style in all contracts, it will make code more obvious.

CLIENT'S COMMENTARY

> Acknowledged - no action will be taken on this codebase, but it will be keep into consideration for future code improvements.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

- Ethereum
- Cosmos
- EOS
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- C++

CONTACTS

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

https://twitter.com/mixbytes