Governor DAO

Vault

Smart Contract Audit Report

October 28, 2021

Introduction 3
About Governor DAO 3
About ImmuneBytes 3
Documentation Details 3
Audit Process & Methodology 4
Audit Details 4
Audit Goals 5
Security Level References 5
High Severity Issues 6
Medium Severity Issues 7
Low Severity Issues 7
Recommendations 11
Unit Test 12
Coverage Report 12
Automated Auditing 12
Contract Library 12
Slither 13
Concluding Remarks 15
Disclaimer 15

Introduction

1. About Governor DAO

Governor DAO is a Wyoming-based Decentralized Autonomous Organization positioned as the “DAO of DAOs”. Governors offer a suite of products and services for projects looking to build out DAO qualities in their own communities. Our offerings include an industry-first sybil-resistance product for one-voice-one-vote governance, as well as governance bootstraps for new communities, consultations, and smart contract porting.

By decentralizing project ownership, Governor Decentralized Autonomous Organization (GDAO) allows founders and core team members to hold less legal liabilities and offer more open based “Sandbox-like" services in DeFi. Driven by its biometrically verified Proof-of-Existence token, GDAO aims to be a role model for other DAO’s by offering the utmost security and anonymity to its end-users in a true, decentralized governance community.

Visit https://www.governordao.org/ to know more about the services.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes’s security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Governor DAO team has provided the following doc for the purpose of audit:

1. https://docs.governordao.org/

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Governor DAO
- Contracts Name: xGDAO
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: 0xA0ab9d44541801e843dD80d11fa9E58B52013b81
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	2	-	8
Closed	-	-	-

High Severity Issues

1. **Contract GDAOBar**

   Wrong calculation of dayCount in leave function.

   Let say the block.timestamp value is 1635182421, depositTimestamp[msg.sender] is 1633999424 and the lock period is 432000

   As per the current calculation in contract the day count = ((1635182421 - 1633999424 + 432000) / 86400) is 18.69 days.

   But if you see the difference between enterTime and leaveTime that is 13.69 days. So we can never calculate the fee for the time which is greater than the difference time.

   Hence the unnecessary adding of lockPeriod makes no sense to the fee calculation.

   To improvise the calculation we suggest two possibilities.
   - The calculation starts after the lock period expires.
   - The calculation starts at the timestamp of entry.

   Calculation starts after the lockPeriod expires

   ```
   uint dayCount = (block.timestamp - depositTimestamp[msg.sender] - lockPeriod) / 86400;
   ```

   Calculation starts at the timestamp of entry

   ```
   uint dayCount = (block.timestamp - depositTimestamp[msg.sender]) / 86400;
   ```

2. **Contract xGDAO, line no 1355**

   ```
   controller = IController(_token);
   ```

   Fee calculation starts at the timestamp of entry

   ```
   controller = IController(_controller);
   ```

   Explanation: the controller is storing the address of _token which is the wrong address assignment and a big typo. It should pass the address of _controller instead. The wrong assignment will raise lots of technical issues and failure of transactions.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. The pragma versions used in some of the contracts are not locked. Consider using the latest versions among 0.6.12 for deploying the contracts and libraries as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

   pragma solidity ^0.6.0; // bad: compiles between 0.6.0 and 0.6.12
   pragma solidity 0.6.0; // good : compiles w 0.6.0 only but not the latest version
   pragma solidity 0.6.12; // best: compiles w 0.6.12

2. In contract ERC20, line 545:
   The constructor definition can be modified
   From:

```
constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;
    }
```

   To:

```
constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
    }
```

   decimals assignment can be moved to line 531 where it is defined

```
uint8 private _decimals = 18;
```

   Explanation: Since we are assigning a fixed value(18) to the `_decimals` variable, we can instead assign it directly outside the constructor. It will reduce the contract size and save gas in initialization. Also, it is good practice to initialize all state variables at the time of declaration for better readability and understanding.

3. In contract GDAOBar, xGDAO
function enter

From:

```
if (totalShares == 0 || totalGDAO == 0) {
            _mint(msg.sender, _amount);
    }
    else {
            uint256 what = _amount.mul(totalShares).div(totalGDAO);
            _mint(msg.sender, what);
    }
```

To:

```
uint256 mintAmount = (totalShares == 0 || totalGDAO == 0) ? _amount :
_amount.mul(totalShares).div(totalGDAO);
_mint(msg.sender, mintAmount);
```

Explanation:“what” is a bad variable name, the _mint call is written twice and the if-else block is used.

We recommend that ternary operations should be opted over if-else and can calculate the minting amount prior to the mint call. This will reduce gas price and contract size.
Also, the naming convention for the variable is bad. The name should opt-in such a manner that it explains the context more explicitly.

4. In contract GDAOBar, in function leave on line no 910, 911

From:

```
if(block.timestamp >= depositTimestamp[msg.sender] + lockPeriod){
            uint dayCount = (block.timestamp -
depositTimestamp[msg.sender] + lockPeriod) / 86400;
```

To:

```
Uint256 senderDepositTimestamp = depositTimestamp[msg.sender]
if(block.timestamp >= senderDepositTimestamp){
            uint dayCount = (block.timestamp -
senderDepositTimestamp) / 86400;
```

Explanation: There are two instances in the contract where the depositTimestamp is read from the storage.

We recommend storing the read to a variable and using that variable in both instances, by this the execution gas gets decreased by **176** and contract deployment gas by **12300**.

5. In contract GDAOBar, in function leave
   From:

```
if(dayCount < 10){
                fee = 10 - dayCount;
            }else{
                fee = 0;
            }
```

   To:

```
if(dayCount < 10){
                fee = 10 - dayCount;
            }
```

   Explanation: The uint256 at the time of declaration with no value gets assigned to 0 and redundant assignment to 0 results in more gas usage. We recommend not to use the else block.

6. Contract GDAOBar, xGDAO line no 905, 1386

   From:

```
if (totalShares == 0 || totalGDAO == 0) {
            _mint(msg.sender, _amount);
        }
        else {
            uint256 what = _amount.mul(totalShares).div(totalGDAO);
            _mint(msg.sender, what);
        }
        GDAO.transferFrom(msg.sender, address(this), _amount);
```

   To:

```
GDAO.transferFrom(msg.sender, address(this), _amount);
if (totalShares == 0 || totalGDAO == 0) {
```

```
            _mint(msg.sender, _amount);
        }
        else {
            uint256 what = _amount.mul(totalShares).div(totalGDAO);
            _mint(msg.sender, what);
        }
```

Explanation: We recommend calling the transferFrom method before the minting process. It results in gas optimization in the case when transferFrom gets reverted and the unnecessary _mint function doesn't get executed.

7. The following function should be declared external as this is not used anywhere within the contract. This saves gas on function call and contract deployment.

   In all public functions, EVM immediately copies array arguments to memory, while external functions can read directly from calldata. Memory allocation is expensive, whereas reading from call data is cheap.

   So we recommend making the "enter" function external in contract GDAOBar and xGDAO.

8. Contract GDAOBar and xGDAO - line no 905, 922, 1393

   Changes Suggested:
   Line no 905 and 1393

```
require(GDAO.transferFrom(msg.sender, address(this), _amount), "GDAO:
Transfer from failed");
```

   Line no 922

```
require(GDAO.transfer(msg.sender, amm.sub(fee)), "GDAO: Transfer
failed");
```

   The transfer/transferFrom return value is unchecked and in this case of failure or returns false. The function call will not fail.

   So we recommend checking the transfer/transferFrom return value and fail in the case of returning false.

Recommendations

1. Add events for most state changes. For example, when setting the setPoeTax, when the controller address gets changed in xGDAO. Events should be fired with all state variable updates as good practice. This makes the contract future proof for usage in frontend applications and event listener backend services.

2. Follow solidity style guide for better readability:
   For example, Functions should be grouped according to their visibility and ordered:
   - constructor
   - receive function (if exists)
   - fallback function (if exists)
   - external
   - public
   - internal
   - private
   - Within a grouping, place the view and pure functions last.

   Note: Linting violations can be easily fixed using linters like solhint.

3. Add Natspecs comments to all functions for a better understanding regarding what the parameters mean and what the function does.

4. All the "*require*" statements used in the contract should also specify error messages for easy debugging.
   Some "require" checks are missing for the input parameter in the xGDAO contract.

5. To improve the readability and consistency use uint256 instead of uint on line no 881 and 911.
6.
7. Contract GDAOBar, line 910, 911
   Contract PoolShareGovernanceToken, line 1080
   Contract xGDAO, line 1382

   We recommend using safe math for the arithmetic operation. Although it is a fairly simple arithmetic operation, safe math adds an extra layer of security over overflow/underflow issues.

Unit Test

No unit tests were provided by the Governor DAO team.

Recommendation:
Our team suggests that the developers should write extensive test cases for the contracts.

Coverage Report

Coverage reports cannot be generated without unit test cases.

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

Automated Auditing

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real-time.

Analysis was performed using the contract Library on the xGDAO contracts on kovan:
Controller contract
0xcd1362c70178265eaa644e48b5c633674ffd011f

Analysis summary can be accessed here:
https://contract-library.com/contracts/Kovan/0xCD1362c70178265eaa644E48b5c633674ffD011F

xGDAO contract
0xF5767CBad80263a63557a837180E280588bEdE83

Analysis summary can be accessed here:
https://contract-library.com/contracts/Kovan/F5767CBad80263a63557a837180E280588bEdE83

It raises a big issue while the controller functions (resetApproval, sweepErc20, approveToken) get called and the transactions fail. As per our investigation in the Constructor of xGDAO, controller address is getting assigned with token Address which raises a lot of technical issues and failure of transactions.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented
static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.

```
GDAOBar.enter(uint256) (contracts/xGDAO.sol#891-906) ignores return value by GDAO.transferFrom(msg.sender,address(this),_amount) (contracts/xGDAO.sol#905)
GDAOBar.leave(uint256) (contracts/xGDAO.sol#908-923) ignores return value by GDAO.transfer(msg.sender,amm.sub(fee)) (contracts/xGDAO.sol#922)
xGDAO.enter(uint256) (contracts/xGDAO.sol#1374-1394) ignores return value by GDAO.transferFrom(msg.sender,address(this),_amount) (contracts/xGDAO.sol#1393)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
PoolShareGovernanceToken._writeCheckpoint(address,uint32,uint256,uint256) (contracts/xGDAO.sol#1127-1144) uses a dangerous strict equality:
   - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/xGDAO.sol#1136)
xGDAO.enter(uint256) (contracts/xGDAO.sol#1374-1394) uses a dangerous strict equality:
   - totalShares == 0 || totalGDAO == 0 (contracts/xGDAO.sol#1385)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
GDAOBar.leave(uint256).fee (contracts/xGDAO.sol#909) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
xGDAO.sweepErc20(address) (contracts/xGDAO.sol#1424-1446) ignores return value by uniswapRouter.swapExactTokensForTokens(amt,1,path,address(this),now + 30) (contracts/xGDAO.sol#1445)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ERC20.constructor(string,string).name (contracts/xGDAO.sol#542) shadows:
   - ERC20.name() (contracts/xGDAO.sol#551-553) (function)
ERC20.constructor(string,string).symbol (contracts/xGDAO.sol#542) shadows:
   - ERC20.symbol() (contracts/xGDAO.sol#559-561) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
xGDAO.constructor(address,address,address,uint8,uint256)._POE (contracts/xGDAO.sol#1351) lacks a zero-check on :
      - POE = _POE (contracts/xGDAO.sol#1357)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
xGDAO._approve(uint256) (contracts/xGDAO.sol#1472-1479) has external calls inside a loop: (pool) = pools.at(i) (contracts/xGDAO.sol#1476)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
GDAOBar.leave(uint256) (contracts/xGDAO.sol#908-923) uses timestamp for comparisons
   Dangerous comparisons:
   - block.timestamp >= depositTimestamp[msg.sender] + lockPeriod (contracts/xGDAO.sol#910)
   - dayCount < 10 (contracts/xGDAO.sol#912)
PoolShareGovernanceToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/xGDAO.sol#1014-1040) uses timestamp for comparisons
   Dangerous comparisons:
   - require(bool,string)(now <= expiry,VSP::delegateBySig: signature expired) (contracts/xGDAO.sol#1038)
xGDAO._beforeTokenTransfer(address,address,uint256) (contracts/xGDAO.sol#1452-1470) uses timestamp for comparisons
   Dangerous comparisons:
   - require(bool,string)(block.timestamp >= depositTimestamp[from].add(lockPeriod),Operation not allowed due to lock period) (contracts/xGDAO.sol#1463-1466)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (contracts/xGDAO.sol#270-279) uses assembly
   - INLINE ASM (contracts/xGDAO.sol#277)
Address._functionCallWithValue(address,bytes,uint256,string) (contracts/xGDAO.sol#363-384) uses assembly
   - INLINE ASM (contracts/xGDAO.sol#376-379)
PoolShareGovernanceToken.getChainId() (contracts/xGDAO.sol#1151-1155) uses assembly
   - INLINE ASM (contracts/xGDAO.sol#1152-1154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
   - Version used: ['0.6.12', '^0.6.0', '^0.6.2', '^0.6.6']
   - ^0.6.0 (contracts/xGDAO.sol#9)
   - ^0.6.0 (contracts/xGDAO.sol#87)
   - ^0.6.2 (contracts/xGDAO.sol#247)
   - ^0.6.0 (contracts/xGDAO.sol#389)
   - ^0.6.0 (contracts/xGDAO.sol#464)
   - ^0.6.0 (contracts/xGDAO.sol#489)
   - ^0.6.0 (contracts/xGDAO.sol#796)
   - 0.6.12 (contracts/xGDAO.sol#864)
   - 0.6.12 (contracts/xGDAO.sol#952)
   - 0.6.12 (contracts/xGDAO.sol#1160)
   - 0.6.12 (contracts/xGDAO.sol#1242)
   - ^0.6.6 (contracts/xGDAO.sol#1272)
   - 0.6.12 (contracts/xGDAO.sol#1301)
   - 0.6.12 (contracts/xGDAO.sol#1333)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (contracts/xGDAO.sol#323-325) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (contracts/xGDAO.sol#348-350) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (contracts/xGDAO.sol#358-361) is never used and should be removed
Address.sendValue(address,uint256) (contracts/xGDAO.sol#297-303) is never used and should be removed
Context._msgData() (contracts/xGDAO.sol#481-484) is never used and should be removed
ERC20._beforeTokenTransfer(address,address,uint256) (contracts/xGDAO.sol#791) is never used and should be removed
ERC20._setupDecimals(uint8) (contracts/xGDAO.sol#773-775) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (contracts/xGDAO.sol#438-441) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (contracts/xGDAO.sol#433-436) is never used and should be removed
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
SafeERC20.safeTransfer(IERC20,address,uint256) (contracts/xGDAO.sol#407-409) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (contracts/xGDAO.sol#411-413) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/xGDAO.sol#223-225) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/xGDAO.sol#239-242) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.6.0 (contracts/xGDAO.sol#9) allows old versions
Pragma version^0.6.0 (contracts/xGDAO.sol#87) allows old versions
Pragma version^0.6.2 (contracts/xGDAO.sol#247) allows old versions
Pragma version^0.6.0 (contracts/xGDAO.sol#389) allows old versions
Pragma version^0.6.0 (contracts/xGDAO.sol#464) allows old versions
Pragma version^0.6.0 (contracts/xGDAO.sol#489) allows old versions
Pragma version^0.6.0 (contracts/xGDAO.sol#796) allows old versions
Pragma version^0.6.6 (contracts/xGDAO.sol#1272) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (contracts/xGDAO.sol#297-303):
   - (success) = recipient.call{value: amount}() (contracts/xGDAO.sol#301)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (contracts/xGDAO.sol#363-384):
   - (success,returndata) = target.call{value: weiValue}(data) (contracts/xGDAO.sol#367)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
xGDAO (contracts/xGDAO.sol#1343-1481) should inherit from IStrategy (contracts/xGDAO.sol#1338-1340)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Parameter GDAOBar.updateLockPeriod(uint256)._newLockPeriod (contracts/xGDAO.sol#887) is not in mixedCase
Parameter GDAOBar.enter(uint256)._amount (contracts/xGDAO.sol#891) is not in mixedCase
Parameter GDAOBar.leave(uint256)._share (contracts/xGDAO.sol#908) is not in mixedCase
Parameter GDAOBar.getPrice(uint256)._share (contracts/xGDAO.sol#927) is not in mixedCase
Variable GDAOBar.GDAO (contracts/xGDAO.sol#873) is not in mixedCase
Contract xGDAO (contracts/xGDAO.sol#1343-1481) is not in CapWords
Parameter xGDAO.setPoeTax(uint8)._new (contracts/xGDAO.sol#1367) is not in mixedCase
Parameter xGDAO.enter(uint256)._amount (contracts/xGDAO.sol#1374) is not in mixedCase
Parameter xGDAO.sweepErc20(address)._erc20 (contracts/xGDAO.sol#1424) is not in mixedCase
Parameter xGDAO.changeController(address)._newController (contracts/xGDAO.sol#1448) is not in mixedCase
Variable xGDAO.POE (contracts/xGDAO.sol#1346) is not in mixedCase
Variable xGDAO.POE_TAX (contracts/xGDAO.sol#1347) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (contracts/xGDAO.sol#482)" inContext (contracts/xGDAO.sol#476-485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
symbol() should be declared external:
   - ERC20.symbol() (contracts/xGDAO.sol#559-561)
decimals() should be declared external:
   - ERC20.decimals() (contracts/xGDAO.sol#576-578)
transfer(address,uint256) should be declared external:
   - ERC20.transfer(address,uint256) (contracts/xGDAO.sol#602-605)
allowance(address,address) should be declared external:
   - ERC20.allowance(address,address) (contracts/xGDAO.sol#610-612)
approve(address,uint256) should be declared external:
   - ERC20.approve(address,uint256) (contracts/xGDAO.sol#621-624)
transferFrom(address,address,uint256) should be declared external:
   - ERC20.transferFrom(address,address,uint256) (contracts/xGDAO.sol#638-642)
increaseAllowance(address,uint256) should be declared external:
   - ERC20.increaseAllowance(address,uint256) (contracts/xGDAO.sol#656-659)
decreaseAllowance(address,uint256) should be declared external:
   - ERC20.decreaseAllowance(address,uint256) (contracts/xGDAO.sol#675-678)
owner() should be declared external:
   - Ownable.owner() (contracts/xGDAO.sol#827-829)
renounceOwnership() should be declared external:
   - Ownable.renounceOwnership() (contracts/xGDAO.sol#846-849)
transferOwnership(address) should be declared external:
   - Ownable.transferOwnership(address) (contracts/xGDAO.sol#855-859)
enter(uint256) should be declared external:
   - GDAOBar.enter(uint256) (contracts/xGDAO.sol#891-906)
   - xGDAO.enter(uint256) (contracts/xGDAO.sol#1374-1394)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Concluding Remarks

While conducting the audits of the Governor DAO smart contract, it was observed that the contracts contain High and Low severity issues.

Our auditors suggest that High and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Governor DAO platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes