HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: xDAO
Date: September 3rd, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for xDAO.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; DAO; DAO Factory; Shop; DAO Viewer
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/xDAO-App/xdao-contracts
Commit	8FEABCFCBF4F03F666448F5500F54E3DAAD3E42E
Technical Documentation	YES
JS tests	YES
Timeline	30 AUGUST 2021 - 03 SEPTEMBER 2021
Changelog	03 SEPTEMBER 2021 - INITIAL AUDIT 03 SEPTEMBER 2021 - SECOND REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by xDAO (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between August 30th, 2021 - September 3rd, 2021. The second code review conducted on September 3rd, 2021.

Scope

The scope of the project is smart contracts in the repository:
Repository:
https://github.com/xDAO-App/xdao-contracts
Commit:
8feabcfcbf4f03f666448f5500f54e3daad3e42e
Technical Documentation: Yes/No?
JS tests: Yes/No?
Contracts:
core/Dao.sol
core/Factory.sol
core/LP.sol
core/Shop.sol
core/XDAO.sol
interfaces/IAdapter.sol
interfaces/IDao.sol
interfaces/IFactory.sol
interfaces/ILP.sol
viewers/DaoViewer.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency

	▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** medium and **8** low severity issues.

After the second review security engineers found **1** medium and **1** low severity issue, which were commented by the customer.

Notice:

1. Quorum, MonthlyCost and FreeTrial values are changed with no events emitting;

2. Minting amount of XDAO tokens is 1 billion.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

No high severity issues were found.

🟩 🟩 Medium

Unused return of a function

The return value of an external call is not stored in a local or state variable. So if the called function will not revert but return false, your execute/executePermitted functions will still return **true** and store the execution.

Recommendation: Please check the return value of called functions.

Customer's comment: *It is normal for us not to store the return value.*

Because we use the Address library from openzeppelin and completely delegate all checks to them.

🟦 Low

1. Implicit visibility declaration

   When visibility is not explicitly declared it is assumed to be internal. But it could be unclear to reviewers.

   Recommendation: Please add an explicit visibility declaration.

   Fixed before the second review.

2. Excess conditions checking

   It is excess to check **uint256** value to be zero or less than the current block **timestamp** because the current block **timestamp** will always be greater than zero.

   Recommendation: Please remove the excess condition check

   Fixed before the second review.

3. Excess require statement

   It is excess to assert if **AddressSet** doesn't contain a value and then adding it because this check is already done in the add function, which will just return false if a value is already in there.

   Recommendation: Please remove the excess require statement.

   Fixed before the second review.

4. Excess require statement

It is excess to assert if **AddressSet** contains a value and then removing it because this check is already done in the remove function, which will just return false if a value is not there.

Recommendation: Please remove the excess require statement.

Fixed before the second review.

5. Unindexed event parameters

   The event DAO.Received doesn't have an indexed field in it. Indexing fields will add the ability to search and better organize logs later.

   Recommendation: Please add **indexed** keyword to the address field of the event.

   Fixed before the second review.

6. Multiple readings for the state variable

   It is more gas sufficient to read the state variable only once and store it to the local variable. Then use the local variable when it's needed in the function.

   Recommendation: Please use local variables.

   Fixed before the second review.

7. Too many digits

   Literals with many digits are difficult to read and review.

   Recommendation: Please use scientific notation and ether units (ie. *1e9 ether*).

Lines: XDAO.sol#16

```
_mint(msg.sender, 1000000000 * 10**decimals());
```

   Customer's comment: *It is normal for us to use this notation. We prefer to keep it.*

8. Missing events

   Changing critical values should be followed by the event emitting to better tracking off-chain.

   Recommendation: Please emit events on changing critical values.

Lines: Dao.sol#488-494

```
function changeQuorum(uint8 _q) external onlyDao returns (bool) {
   require(_q >= 1 && _q <= 100, "DAO: quorum should be 1 <= q <= 100");

   quorum = _q;

   return true;
}
```

Lines: Factory.sol#46-50

```
function changeMonthlyCost(uint256 _m) external onlyOwner returns (bool)
{
   monthlyCost = _m;

   return true;
}
```

Lines: Factory.sol#52-60

```
function changeFreeTrial(uint256 _freeTrial)
   external
   onlyOwner
   returns (bool)
{
   freeTrial = _freeTrial;

   return true;
}
```

Customer's comment: *It's normal for us to not use Events here.We prefer to keep it without Events.*

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **1** medium and **8** low severity issues.

After the second review security engineers found **2** low severity issues, which were commented by the customer.

Notice:

1. Quorum, MonthlyCost and FreeTrial values are changed with no events emitting;

2. Minting amount of XDAO tokens is 1 billion.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

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