HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: XETA Capital
Date: October 27th, 2022

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 XETA Capital
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token
Platform	EVM
Network	Avalanche
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.xetacapital.com/
Timeline	20.09.2022 - 27.10.2022
Changelog	27.09.2022 - Initial Review 20.10.2022 - Second Review 27.10.2022 - Third Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by XETA Capital (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 contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/bpxc/xc-smart-contract-v1
Commit:
01f2785bf3e8671c3dd2e33c5dddffe0ea440aba
Documentation: No

Integration and Unit Tests: No
Deployed Contracts Addresses:
https://snowtrace.io/address/0x827eb4bada6cb76c90f887969b3fe5fad585ffe3#code
Contracts:
File: ./xetatoken.sol
SHA3: d4ab5980c099cf168710e571ea0a3aa77707f67c40e2cf3bdf0e941636d6056a

Second review scope

Repository:
https://gitlab.com/xetapublic/xetasmartcontract
Commit:
3e66d4a5ec7def7a74587b2df7b657e0142448cd
Documentation:
https://gitlab.com/xetapublic/xetasmartcontract/-/blob/main/README.md

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./xetatoken.sol
SHA3: 055149e9f40923d42914b889cd8da1b67df1bfa831835a646a3978f3574e5a5f

Third review scope

Repository:
https://gitlab.com/xetapublic/xetasmartcontract
Commit:
c4febee5c08d2cc498fe13f5b639896ec3b6c67e
Documentation:
https://gitlab.com/xetapublic/xetasmartcontract/-/blob/main/README.md

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./xetatoken.sol
SHA3: 9c6c57e098e24d85c73b6bea11b5121b1678b31eea33cbf7d7e3a6cd9cd3922e

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

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 cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical descriptions are provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The code partially follows official language style guides, and tests are not functioning.

Test coverage

Test coverage of the project is **100.00%**.

Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**.

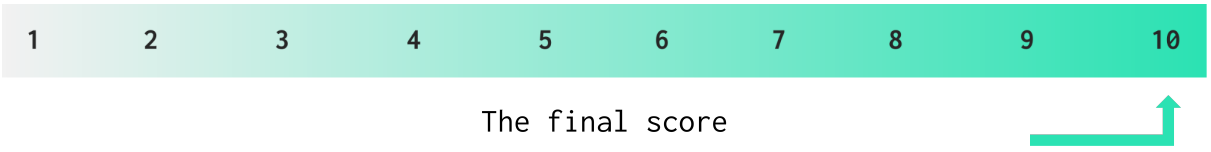

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 September 2022	6	4	6	0
18 October 2022	3	3	2	0
26 October 2022	2	0	0	0

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

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

XETA - ERC20 token that mints all initial supply to own address *with the logic inherited from OpenZeppelin*
contracts. Additional minting and pausing of transfers are allowed.

It has the following attributes:

- Name: XETA
- Symbol: XETA
- Decimals: 18
- Initial supply: 21m tokens.

Privileged roles

- The owner of the *XETA* contract can pause/unpause the contract, change fee and token in which fee will be paid, withdraw any token from the contract, as well as XETA tokens which were initially minted.
- The Xeta wallet can set a fee for rewards claims and transfer tokens from the contract address to the chosen user.

Risks

- Pause the token transfers.
- Fee for system operations can be changed at any time.

Findings

Critical

No critical severity issues were found.

High

1. **Funds Lock**

   XetaToken has a *receive()* function which allows sending native chain currency but does not have a function to withdraw Ether.

   Native coins can be locked.

   Path: ./xetatoken.sol

   Recommendation: Remove the receive method from the contract.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

2. **Undocumented Behavior**

   The owner can stop all the token transfers with a pause function.

   This can lead to the users' funds manipulation.

   Path: ./xetatoken.sol

   Function: pause

   Recommendation: Remove pausing functionality or describe owner permissions in the documentation.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

3. **Token Supply Manipulation**

   The owner has permission to mint an unlimited supply of tokens.

   It can lead to token price manipulation.

   Path: ./xetatoken.sol

   Function: mintTokens

   Recommendation: Limit the number of tokens that can be minted and add NatSpec for purpose and restriction of minting new tokens.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

4. **Requirements Violation**

   It’s possible for the owner to change the main payment token from USDC to any desired one.

   It can lead to a loss of accidentally allowed tokens by the user.

   Path: ./xetatoken.sol

Function: setUsdcToken

Recommendation: Remove the possibility of changing the main payment token or provide documentation with a description of why this is desired. Move the setting of the payment token to the constructor.

Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

5. **Invalid Calculations**

The fee for user actions is distributed between 3 wallets according to the percentage settled by the contract owner. It is possible to set the total percentage of fees to be greater than 100%.

This will lead to the user paying a higher commission than expected.

Path: ./xetatoken.sol

Function: setWalletDistribution

Recommendation: Add additional checks to prevent exceeding 100% percentage distribution.

Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

6. **Instant Contract Parameters Change**

Change of variables without proper notification and delay before applying new values can lead to unexpected fees paid by the user.

Path: ./xetatoken.sol

Functions: setXonCost, setXonCreationFee, setSubscriptionFee, setClaimRewardFee

Recommendation: Add the fee amount as a parameter to methods requestRewards, mintXon, mintXonFromXpre, requestSubscriptions and then check if it is the same as required. Users will be safe from an unexpected change of the fee.

Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

Medium

1. **Unchecked Return Value**

The return value of an external *transfer* and *transferFrom* call is not checked.

Path: ./xetatoken.sol

Functions: distributePaymentInUsdc, withdrawToken

Recommendation: Implement a check of the returning value.

Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

2. **Missing Events Emit on Changing Important Values**

   The contract does not emit any events after changing important values.

   Path: ./xetatoken.sol

   Functions: setXetaWallet, setXonCost, setXonCreationFee, setSubscriptionFee, setClaimRewardFee, setMaxXonsPerUser, setUsdcToken, setWalletDistribution, setDistributedWallet

   Recommendation: Implement event emits after changing contract values.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

3. **Tautology of Contradiction**

   The code contains the following validation: *number >=0*. Though, uint value is always >=0. Used validation is redundant.

   Path: ./xetatoken.sol

   Functions: getWalletDistribution, getDistributedWallet, setWalletDistribution, setDistributedWallet

   Recommendation: Fix the incorrect comparison.

   Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

4. **Lack of Validation for the Function Parameter**

   If zero amount fee is settled, the contract will continue to perform USDC transfers to distributed wallets.

   Path: ./xetatoken.sol

   Function: distributePaymentInUsdc

   Recommendation: Add *if* statement to check if it is needed to perform the token transfer.

   Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

Low

1. **Floating Pragma**

   Unlocked pragmas may cause the contract to be deployed with a different Solidity version from the tested. The project uses floating pragmas >=0.8.12.

   Path: ./xetatoken.sol

   Recommendation: Lock pragma to a specific compiler version.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

2. **Usage of State Variables Default Visibility**

   Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

   Path: ./xetatoken.sol

   Recommendation: Explicitly define visibility for all state variables.

   Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

3. **Functions that Could Be Declared External**

   Public functions that are never called by the contract should be declared external to save Gas.

   Path: ./xetatoken.sol

   Functions: pause, unpause,setXetaWallet,setXonCost,setXonCreationFee, setSubscriptionFee,setClaimRewardFee,setMaxXonsPerUser, setUsdcToken, getUsdcToken,getXonCost,getXonCreationFee,getClaimRewardFee, getSubscriptionFee,getWalletDistribution,getDistributedWallet, getMaxXonsPerUser, setWalletDistribution, setDistributedWallet

   Recommendation: use the external attribute for functions never called from the contract.

   Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

4. **Unindexed Events**

   Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

   Path: ./xetatoken.sol

   Recommendation: Use the “indexed” keyword to the event parameters.

   Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

5. **No Zero Address Validation**

   Address parameters are being used without checking against the possibility of 0x0.

   Path: ./xetatoken.sol

   Functions: setXetaWallet, setUsdcToken, setDistributedWallet

   Recommendation: Add zero address validation.

   Status: Fixed (c4febee5c08d2cc498fe13f5b639896ec3b6c67e)

6. **Style Guide Violation**

   The provided projects do not follow the official guidelines.

   Path: ./xetatoken.sol

   Recommendation: Follow the official Solidity guidelines.

Status: Fixed (3e66d4a5ec7def7a74587b2df7b657e0142448cd)

7. **Redundant Require Statement**

   In function *setDistributedWallet* two require statements with check of 0x0 for one function parameter.

   Path: ./xetatoken.sol

   Function: setDistributedWallet

   Recommendation: Remove redundant require statement.

   Status: New

8. **Redundant if Statement**

   After changes, function *distributePaymentInUsdc* will be called only if there is a positive amount of fees. Check for *amount == 0* is redundant.

   Path: ./xetatoken.sol

   Function: distributePaymentInUsdc

   Recommendation: Remove redundant if statement.

   Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.