Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Synthetix Network Token
Website:	synthetix.io
Platform:	Ethereum
Language:	Solidity
Date:	May 6th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of Synthetix Network Token from synthetix.io were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 6th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Synthetix Network code consists of three primary contracts: Owned, Proxy, and Proxyable, along with an interface IERC20 and a contract ProxyERC20 implementing the ERC20 interface.
- Here's a summary of each contract:
    - **Owned.sol:** This contract provides basic ownership functionality. It allows for setting and changing the owner of the contract through a nomination process. Only the current owner can nominate a new owner, and the nominated owner must accept the nomination.
    - **Proxy.sol:** This contract acts as a proxy for another contract (Proxyable). It can toggle between using DELEGATECALL and CALL to execute functions on the target contract. If a function is not recognized by the proxy, it forwards the call to the target contract. The proxy's functionality can be controlled by the owner.
    - **Proxyable.sol:** This contract is designed to work with Proxy. It allows for interaction with the underlying contract through the proxy. It includes modifiers to ensure that only the proxy can call certain functions.
    - **ProxyERC20.sol:** This contract implements the ERC20 interface and acts as a proxy specifically for ERC20 token contracts. It forwards ERC20 function calls to the target contract. It ensures that contract-to-contract calls are compatible with the ERC20 standard.

- Overall, these contracts provide a framework for creating upgradeable contracts using proxies, with ownership control and support for ERC20 token functionality.

Audit scope

Name	Code Review and Security Analysis Report for Synthetix Network Token Smart Contract
Platform	**Ethereum**
File	ProxyERC20.sol
Smart Contract Code	0xc011a73ee8576fb46f5e1c5751ca3b9fe0af2a6f
Audit Date	May 6th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Synthetix Network Token • Symbol: SNX • Decimals: 18	**YES, This is valid.**
Ownership control: • Proxy addresses can be set by the owner. • Integration Proxy address can be set by the owner. • Message Sender address can be set by the Proxy owner. • Target can be set by the owner. • Delegate calls can be set by the owner. • Emit by the proxy target owner. • Nominate a new owner by the current owner. • Accept the nomination to be owned by the current owner.	**YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 7 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	No
Is it a Proxy?	Yes
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well written smart contract.

The libraries in Synthetix Network Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Synthetix Network Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Synthetix Network Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Incorrect Placement of Visibility Modifier	Refer Audit Findings
2	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	allowance	read	Passed	No Issue
8	transfer	write	Passed	No Issue
9	approve	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	allowance	read	Passed	No Issue
14	transfer	write	Unchecked Transfer	Refer Audit Findings
15	approve	write	Passed	No Issue
16	transferFrom	write	Unchecked Transfer	Refer Audit Findings
17	name	read	Passed	No Issue
18	symbol	read	Passed	No Issue
19	decimals	read	Passed	No Issue
20	setProxy	external	Centralized risk	Refer Audit Findings
21	setIntegrationProxy	external	Centralized risk	Refer Audit Findings
22	setMessageSender	external	Centralized risk	Refer Audit Findings
23	onlyProxy	modifier	Passed	No Issue
24	optionalProxy	modifier	Passed	No Issue
25	optionalProxy_onlyOwner	modifier	Passed	No Issue
26	setTarget	external	Centralized risk, Missing Zero Address Validation	Refer Audit Findings
27	setUseDELEGATECALL	external	Centralized risk	Refer Audit Findings
28	_emit	external	access only Target	No Issue
29	onlyTarget	modifier	Passed	No Issue
30	nominateNewOwner	external	Centralized risk, Missing Zero Address Validation	Refer Audit Findings
31	acceptOwnership	external	Passed	No Issue
32	onlyOwner	modifier	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: **Owned.sol**

```
41    pragma solidity 0.4.25;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.7.

(2) Error message for require is too long: **Owned.sol**

```
require(msg.sender == nominatedOwner, "You must be nominated before you can accept ownership");
```

Ethereum has a gas limit for each block. This limit includes the gas used by all transactions and contract executions within that block. When a required statement fails, it results in an exception, and the error message, along with the gas used up to that point, is included in the transaction's revert message.

Resolution: We suggest writing short and clear messages in the required statements.

(3) Centralized risk:

Owned.sol

```
function nominateNewOwner(address _owner)
    external
    onlyOwner
{
    nominatedOwner = _owner;
    emit OwnerNominated(_owner);
}
```

Owned.sol

- onlyOwner can call a nominateNewOwner function.

Proxy.sol

- onlyOwner can call the setTarget function and setUseDELEGATECALL function.
- onlyTarget can call _emit function.

Proxyable.sol

- onlyOwner can call the setProxy function, setIntegrationProxy function.
- onlyProxy can call the setMessageSender function.

Resolution: To make the smart contract 100% decentralized. We suggest renouncing ownership of the smart contract once its function is completed.

(4) Missing Zero Address Validation:

Owned.sol

```
function nominateNewOwner(address _owner)
    external
    onlyOwner
{
    nominatedOwner = _owner;
    emit OwnerNominated(_owner);
}
```

- nominateNewOwner

Proxy.sol

- setTarget

Some functions in this contract may not appropriately check for zero addresses being used.

Resolution: check that the address is not zero.

(5) Incorrect Placement of Visibility Modifier:

Proxy.sol

```
constructor(address _owner)
    Owned(_owner)
    public
{}
```

Proxyable.sol

```
constructor(address _proxy, address _owner)
    Owned(_owner)
    public
{
    proxy = Proxy(_proxy);
    emit ProxyUpdated(_proxy);
}
```

ProxyERC20.sol

```
constructor(address _owner)
    Proxy(_owner)
    public
{}
```

During the code review, it was observed that the constructor in the smart contract contains a visibility modifier (public) that is placed after other modifiers and the constructor's base constructor call. According to Solidity's syntax conventions, visibility modifiers should be the first modifiers in the list.

Resolution: We recommend adhering to Solidity's syntax guidelines by moving the visibility modifier to the beginning of the list of modifiers.

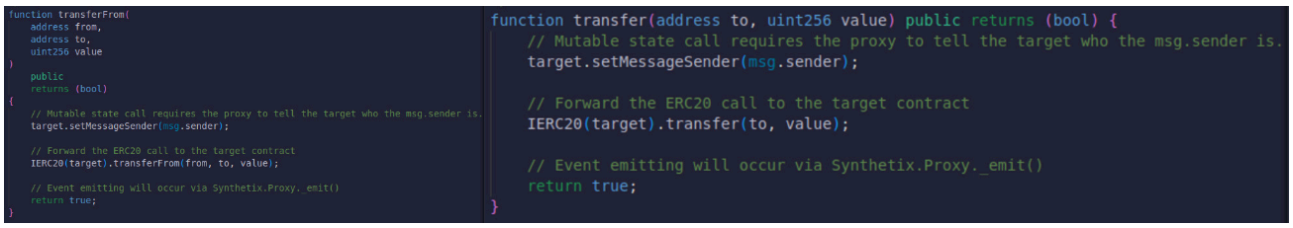
(6) Explicit Visibility for State Variables Warning: **Proxyable.sol**

```
/* The caller of the proxy, passed through to this contract.
 * Note that every function using this member must apply the onlyProxy or
 * optionalProxy modifiers, otherwise their invocations can use stale values. */
address messageSender;
```

The warning is related to the visibility of state variables in your Solidity code.

Resolution: We recommend updating the code to explicitly mark the visibility of state variables using the internal or public keyword, depending on the intended visibility.

(7) Unchecked Transfer: **ProxyERC20.sol**



```
function transferFrom(
    address from,
    address to,
    uint256 value
)
    public
    returns (bool)
{
    // Mutable state call requires the proxy to tell the target who the msg.sender is.
    target.setMessageSender(msg.sender);

    // Forward the ERC20 call to the target contract
    IERC20(target).transferFrom(from, to, value);

    // Event emitting will occur via Synthetix.Proxy._emit()
    return true;
}
```

```
function transfer(address to, uint256 value) public returns (bool) {
    // Mutable state call requires the proxy to tell the target who the msg.sender is.
    target.setMessageSender(msg.sender);

    // Forward the ERC20 call to the target contract
    IERC20(target).transfer(to, value);

    // Event emitting will occur via Synthetix.Proxy._emit()
    return true;
}
```

A transfer call made in this contract may be unstable and cause tokens to become stuck.

Resolution: Review and optimize the transfer function to handle potential exceptions and ensure it doesn't leave tokens stuck. Implement proper error handling and checks to prevent instability.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

Proxyable.sol

- setProxy: Proxy address can be set by the owner.
- setIntegrationProxy: The Integration Proxy address can be set by the owner.
- setMessageSender: Message Sender address can be set by the Proxy owner.

Proxy.sol

- setTarget: Target can be set by the owner.
- setUseDELEGATECALL: Delegate call can be set bythe owner.
- _emit: Emit by the proxy target owner.

Owned.sol

- nominateNewOwner: Nominate a new owner by the current owner.
- acceptOwnership: Accept the nomination to be owned by the current owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 7 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

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

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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).

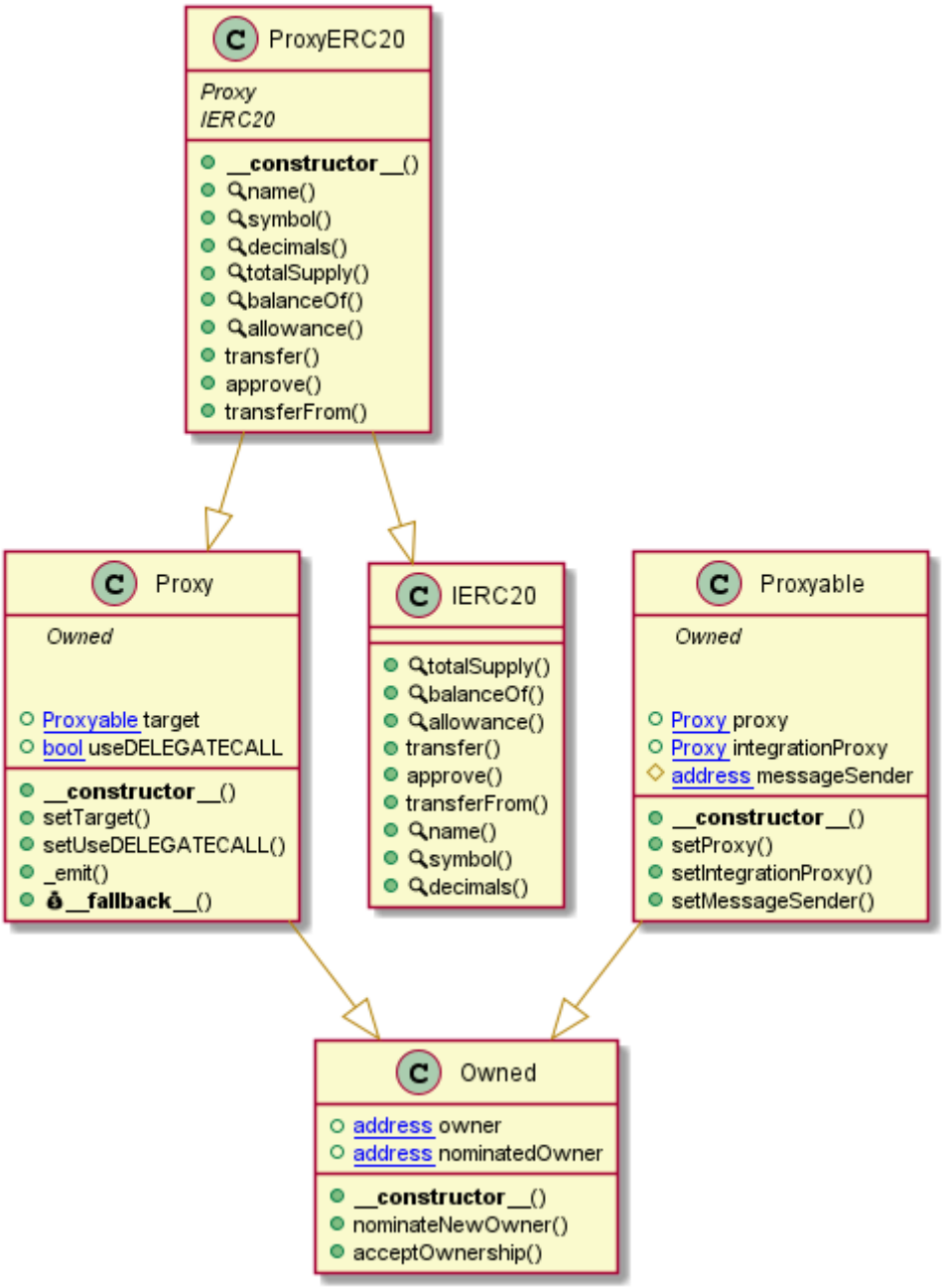
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

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

Appendix

Code Flow Diagram - Synthetix Network Token



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> ProxyERC20.sol

```
ProxyERC20.transferFrom(address,address,uint256) (ProxyERC20.sol#364-380) uses arbitrary from in transferFrom: IERC20(target).
transferFrom(from,to,value) (ProxyERC20.sol#376)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

ProxyERC20.transfer(address,uint256) (ProxyERC20.sol#327-336) ignores return value by IERC20(target).transfer(to,value) (Proxy
ERC20.sol#332)
ProxyERC20.transferFrom(address,address,uint256) (ProxyERC20.sol#364-380) ignores return value by IERC20(target).transferFrom(
from,to,value) (ProxyERC20.sol#376)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Contract locking ether found:
        Contract ProxyERC20 (ProxyERC20.sol#260-382) has payable functions:
         - Proxy.fallback() (ProxyERC20.sol#116-150)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

ProxyERC20.approve(address,uint256) (ProxyERC20.sol#347-356) ignores return value by IERC20(target).approve(spender,value) (Pr
oxyERC20.sol#352)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

ProxyERC20.balanceOf(address).owner (ProxyERC20.sol#299) shadows:
        - Owned.owner (ProxyERC20.sol#11) (state variable)
ProxyERC20.allowance(address,address).owner (ProxyERC20.sol#311) shadows:
        - Owned.owner (ProxyERC20.sol#11) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Owned.nominateNewOwner(address)._owner (ProxyERC20.sol#29) lacks a zero-check on :
                - nominatedOwner = _owner (ProxyERC20.sol#33)
Proxyable.setMessageSender(address).sender (ProxyERC20.sol#195) lacks a zero-check on :
                - messageSender = sender (ProxyERC20.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
Proxy._emit(bytes,uint256,bytes32,bytes32,bytes32,bytes32) (ProxyERC20.sol#84-114) uses assembly
        - INLINE ASM (ProxyERC20.sol#91-114)
Proxy.fallback() (ProxyERC20.sol#116-150) uses assembly
        - INLINE ASM (ProxyERC20.sol#121-134)
        - INLINE ASM (ProxyERC20.sol#138-149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Pragma version0.4.25 (ProxyERC20.sol#3) allows old versions
solc-0.4.25 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Owned.nominateNewOwner(address)._owner (ProxyERC20.sol#29) is not in mixedCase
Parameter Proxy.setTarget(Proxyable)._target (ProxyERC20.sol#69) is not in mixedCase
Function Proxy._emit(bytes,uint256,bytes32,bytes32,bytes32,bytes32) (ProxyERC20.sol#84-114) is not in mixedCase
Parameter Proxyable.setProxy(address)._proxy (ProxyERC20.sol#180) is not in mixedCase
Parameter Proxyable.setIntegrationProxy(address)._integrationProxy (ProxyERC20.sol#188) is not in mixedCase
Modifier Proxyable.optionalProxy_onlyOwner() (ProxyERC20.sol#215-222) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
ProxyERC20.sol analyzed (5 contracts with 84 detectors), 19 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

ProxyERC20.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Proxy.(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 187:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 162:8:

Gas costs:

Fallback function of contract ProxyERC20 requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.

Pos: 187:4:

Constant/View/Pure functions:

Proxy.(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 135:4:

Email: audit@EtherAuthority.io

Constant/View/Pure functions:

ProxyERC20.(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 390:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 81:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 317:8:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

ProxyERC20.sol

```
Compiler version 0.4.25 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:40
Error message for require is too long
Pos: 9:80
Error message for require is too long
Pos: 9:88
Visibility modifier must be first in list of modifiers
Pos: 9:136
Code contains empty blocks
Pos: 5:137
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:161
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:191
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:208
Explicitly mark visibility of state
Pos: 5:266
Visibility modifier must be first in list of modifiers
Pos: 9:270
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io