EtherAuthority

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer:	XDriveToken
Prepared on:	03/12/2020
Platform:	Ethereum
Language:	Solidity

audit@etherauthority.io

Table of contents

Document

Name	Smart Contract Code Review and Security Analysis Report for XDriveToken
Platform	Ethereum / Solidity
MD5 hash	CC774C8A753764685D79B4FC45541D52
File name	XDriveToken.sol
SHA256 hash	EACA8539BCB1824A3A5188284CB133B3152448811E10ACBC8770C5241BC0B8C2
Date	03/12/2020

Introduction

EtherAuthority.io (Consultant) was contracted by XDrive Token Team (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between November 26th, 2020 – December 3rd, 2020.

Project Scope

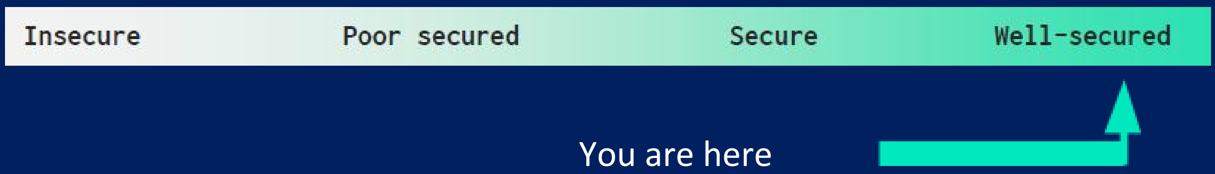
The scope of the project is XDrive Token smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Customer`s solidity smart contract is well secured.



Our team performed analysis of code functionality, manual audit and automated checks with smartDec, Mythril, Slither and remix IDE. All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all found issues can be found in the Audit overview section.

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

Code Quality

XDriveToken protocol consists of one smart contract file.This single file smart contracts also contains safeMath and toAddress library, It is a compact and well written contract.

The libraries in the XDriveToken protocol 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 XDriveToken protocol.

XDriveToken team has **not** provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost not commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

We were given an XDriveToken contract in the form of a file. The hash of that file is mentioned above in the table.

Comments are very helpful in understanding the overall architecture of the protocol. It also provided a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects. And even core code blocks are written well and systematically.

AS-IS overview

XDrivenToken **contract overview**

XDrivenToken is a smart contract that provides a standard mechanism to grow capital via demand supply control mechanism for balanced maximum appreciation on investment.

Contract for audit.sol

Contract: XDrivenToken

Inherit: null

About: This contract provides a good earning setup with secured balanced growth.

Observation: Passed

Test Report: All passed including security check.

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addcontract	write	Passed	All Passed	No Issue	Passed
2	setAdmin	write	Passed	All Passed	No Issue	Passed
3	getAdminAmount	read	Passed	All Passed	No Issue	Passed
4	getAdminPayments	read	Passed	All Passed	No Issue	Passed
5	buy	read	Passed	All Passed	No Issue	Passed
6	reinvest	read	Passed	All Passed	No Issue	Passed
7	exit	write	Passed	All Passed	No Issue	Passed
8	withdraw	write	Passed	All Passed	No Issue	Passed
9	sell	write	Passed	All Passed	No Issue	Passed
10	transfer	write	Passed	All Passed	No Issue	Passed
11	totalEtherumBalance	read	Passed	All Passed	No Issue	Passed
12	totalsupply	read	Passed	All Passed	No Issue	Passed
13	myTokens	read	Passed	All Passed	No Issue	Passed
14	mydividends	read	Passed	All Passed	No Issue	Passed
15	balanceof	read	Passed	All Passed	No Issue	Passed
16	dividendof	read	Passed	All Passed	No Issue	Passed
17	sellprice	read	Passed	All Passed	No Issue	Passed
18	buyprice	read	Passed	All Passed	No Issue	Passed
19	CalculateTokenReceived	read	Passed	All Passed	No Issue	Passed
20	CalculateEthereumReceived	read	Passed	All Passed	No Issue	Passed
21	purchaseTokens	write	Passed	All Passed	No Issue	Passed
22	ethereumToTokens_	read	Passed	All Passed	No Issue	Passed
23	tokensToEthereum_	read	Passed	All Passed	No Issue	Passed
24	sqrt	read	Passed	All Passed	No Issue	Passed
25	getReferralBalance	read	Passed	All Passed	No Issue	Passed
26	getPayouts	read	Passed	All Passed	No Issue	Passed
27	ProfitPerShare	read	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens 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 functions
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.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

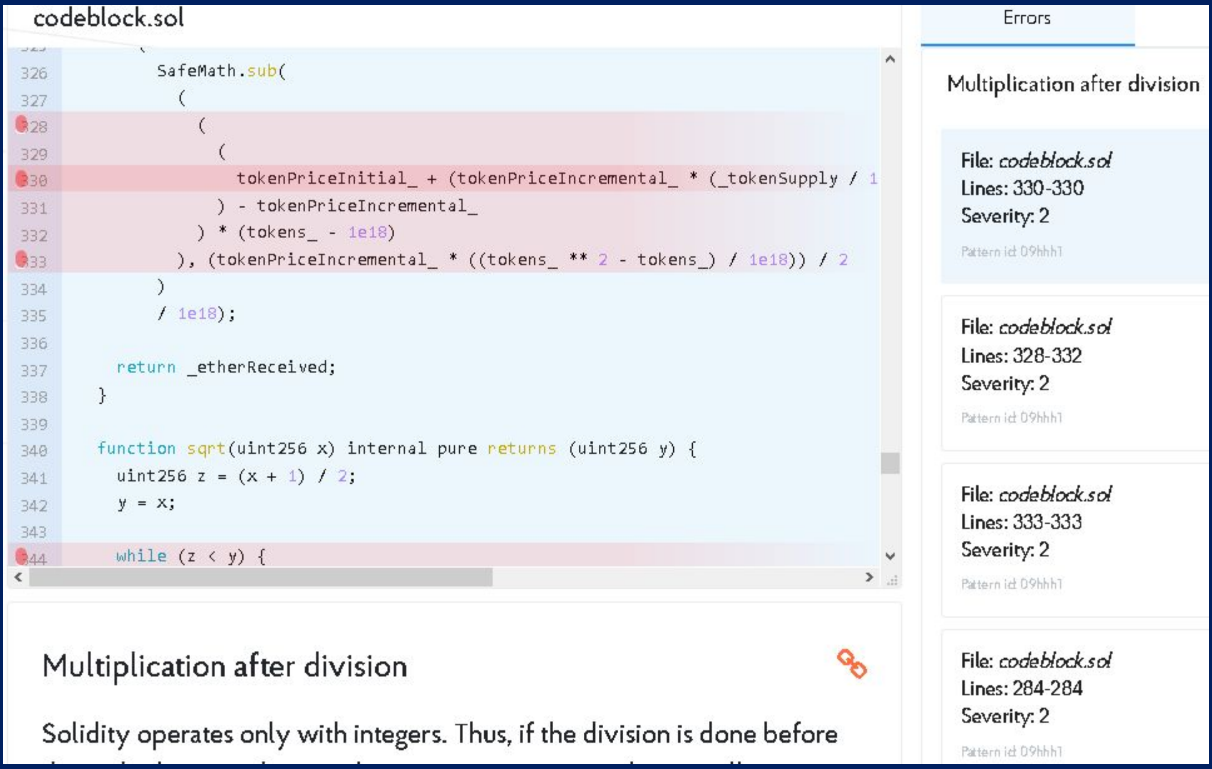
Low

(1) SafeMath is missing in some places. Although it costs a very little extra gas, it protects from all overflow/underflow scenarios, which may be overlooked by the developer.

Very Low

(1) Compiler version is not fixed but after compiling it has no effect on generated machine code. but the lower compiler version is little vulnerable to some compiler oriented bugs (like memory overlap on fixed size arrays), which normally are fixed on later versions. so it is good to use the latest stable version to be on the safe side.

(2) Static tools show the following error but none of this is valid under use cases.



codeblock.sol

```
326          SafeMath.sub(
327            (
328              (
329                (
330                  tokenPriceInitial_ + (tokenPriceIncremental_ * (_tokenSupply / 1
331                ) - tokenPriceIncremental_
332              ) * (tokens_ - 1e18)
333            ), (tokenPriceIncremental_ * ((tokens_ ** 2 - tokens_) / 1e18)) / 2
334          )
335          / 1e18);
336
337      return _etherReceived;
338    }
339
340    function sqrt(uint256 x) internal pure returns (uint256 y) {
341      uint256 z = (x + 1) / 2;
342      y = x;
343
344      while (z < y) {
```

Multiplication after division

Solidity operates only with integers. Thus, if the division is done before

Errors

Multiplication after division

File: *codeblock.sol*
Lines: 330-330
Severity: 2
Pattern id: 09hhh1

File: *codeblock.sol*
Lines: 328-332
Severity: 2
Pattern id: 09hhh1

File: *codeblock.sol*
Lines: 333-333
Severity: 2
Pattern id: 09hhh1

File: *codeblock.sol*
Lines: 284-284
Severity: 2
Pattern id: 09hhh1

Conclusion

We were given contract files. And we have used all possible tests based on given objects as files. The contracts are written so systematic (but not commented), that we found no critical things. So it is good to go for production after adding safeMath for enhanced security.

Since possible test cases can be unlimited for such extensive smart contract protocol, so we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover 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 As-is overview section of the report.

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

Security state of reviewed contract is “Well 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

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).

Because the total number of test cases are unlimited, so 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 recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on 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.

Ether Authority