SHELLBOXES

StartFi

Smart Contract Security Audit

Prepared by: ShellBoxes

August 6, 2021 – August 12, 2021

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	StartFi
Target	StartFi Token Smart Contract
Version	1.0
Auditors	Farouk El Alem, Inas Hasnaoui
Reviewed By	Inas Hasnaoui
Approved By	Inas Hasnaoui
Classification	Public

Document History

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Initial Draft	August 6, 2021	Farouk El Alem
0.2	Additional Findings	August 8, 2021	Inas Hasnaoui
0.3	Final Version	August 11, 2021	Inas Hasnaoui
1.0	Remediation Plan	August 12, 2021	Farouk El Alem

Contacts

VERSION	COMPANY	EMAIL
Inas Hasnaoui	ShellBoxes	i.hasnaoui@shellboxes.com
Farouk El Alem	ShellBoxes	f.elalem@shellboxes.com

Contents

1

1. Introduction

StartFi engaged ShellBoxes to conduct a security assessment on the StartFiToken Smart Contract beginning on August 6th, 2021 and ending August 12th, 2021. We detail our methodical methodology in this report to evaluate potential security issues in the smart contract implementation, exposing possible semantic discrepancies between the smart contract code and design document, and making additional ideas or recommendations for improvement. Our findings indicate that the current version of the smart contract can be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1. About StartFi

StartFI is an NFT Platform to Help Content Creators Raise Funds for Their Digital Content, Engaging Community to Share Rewards & Revenues.

Issuer	StartFi
Website	https://startfi.io
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2. Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to strike a balance between efficiency, timeliness, practicability, and correctness in relation to the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect items that violate security best practices.

1.2.1. Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

1.3. Scope

The StartFiToken Contract in the StartFi Repository

Commit ID: ce6bcaa3f9255743236f3db530f0f1f9e1ed57ca

2. Findings Overview

2.1. Summary

The following is a synopsis of our conclusions from our analysis of the StartFi implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2. Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 1 medium-severity vulnerability, 1 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Approve Race Condition	Medium	Acknowledged
Usage Of Block.TimeStamp	Low	Acknowledged
Floating Pragma	Informational	-
Use of Inline Assembly	Informational	-
Use of Nonces	Informational	-

3. Findings Details

3.1. Approve Race Condition [MEDIUM]

Description:

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 1 : StartFiToken (Line 62)

```
        require(
            verifyEIP712(target, hashStruct, v, r, s) ||
                verifyPersonalSign(target, hashStruct, v, r, s)
        );
        _approve(target, spender, value);
    }
```

Risk Level:

Likelihood – 2

Impact - 5

Recommendation:

Override the _approve function to use the following definition.

Fix 1 : StartFiToken

```
    function _approve(address delegate, uint256 _currentValue, uint256 numTokens) public
override returns (bool) {

      if(_currentValue == allowed[msg.sender][delegate])

      {

         allowed[msg.sender][delegate] = numTokens;

         emit Approval(msg.sender, delegate, numToken);

         return true;

      }

      else return false

}
```

Risk Acknowledged:

The StartFi team acknowledged the risk and chose to use the increaseAllowance and decreaseAllowance to change the approval amount instead of overwriting it using the approve function.

3.2. Usage of Block.TimeStamp [LOW]

Description:

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 2 : StartFiToken (Line 50)

```
    require(block.timestamp <= deadline, "AnyswapV3ERC20: Expired permit");
```

Listing 3 : StartFiToken (Line 74)

```
    require(block.timestamp <= deadline, "StartFiToken: Expired permit");
```

Risk Level:

Likelihood – 3

Impact - 2

Recommendation:

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Risk accepted:

The StartFi team accepted the risk and preferred not to use the oracle for the reason that 900 seconds won't have an impact on the business logic.

3.3. Floating Pragma [Informational]

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Code:

Listing 5 : StartFiToken (Lines 3, 4)

```
pragma solidity >=0.8.0;
pragma experimental SMTChecker;
```

Risk Level:

Likelihood – 2

Impact – 1

Recommendation:

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Remediation:

The StartFi team have locked the pragma version in the config files.

3.4. Use of Inline Assembly [Informational]

Description:

Inline assembly is a way to access the EVM at a low level. This discards several important safety features in Solidity.

Code:

Listing 6 : StartFiToken (Lines 3, 4)

```
        uint chainId;
        assembly {
            chainId := chainId
        }
```

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM at a low level. An attacker could bypass many important safety features of Solidity.

3.5. Use of Nonces [Informational]

Description:

The mapping nonces registers how many signatures have been used for a particular holder. When creating the signature, a nonces value needs to be included. When executing permit, the nonce included must exactly match the number of signatures that have been used so far for that holder. This ensures that each signature is used only once.

All these three conditions together, the PERMIT_TYPEHASH, the DOMAIN_SEPARATOR, and the nonce, make sure that each signature is used only for the intended contract, the intended function, and only once.

Code:

Listing 7 : StartFiToken (Lines 3, 4)

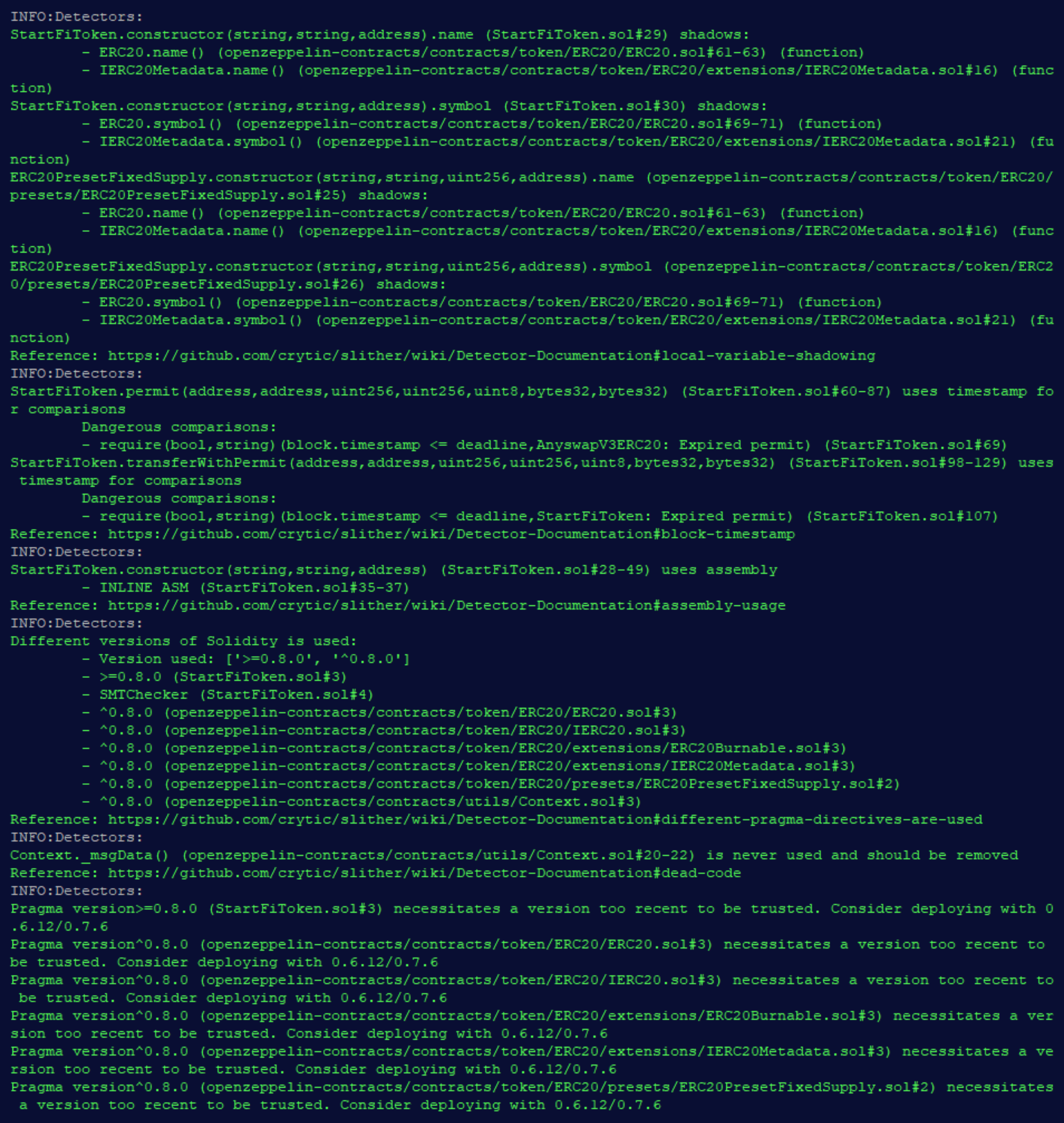
```
    mapping (address => uint256) public  nonces;
    constructor(string memory name,
```

3.6. Static Analysis

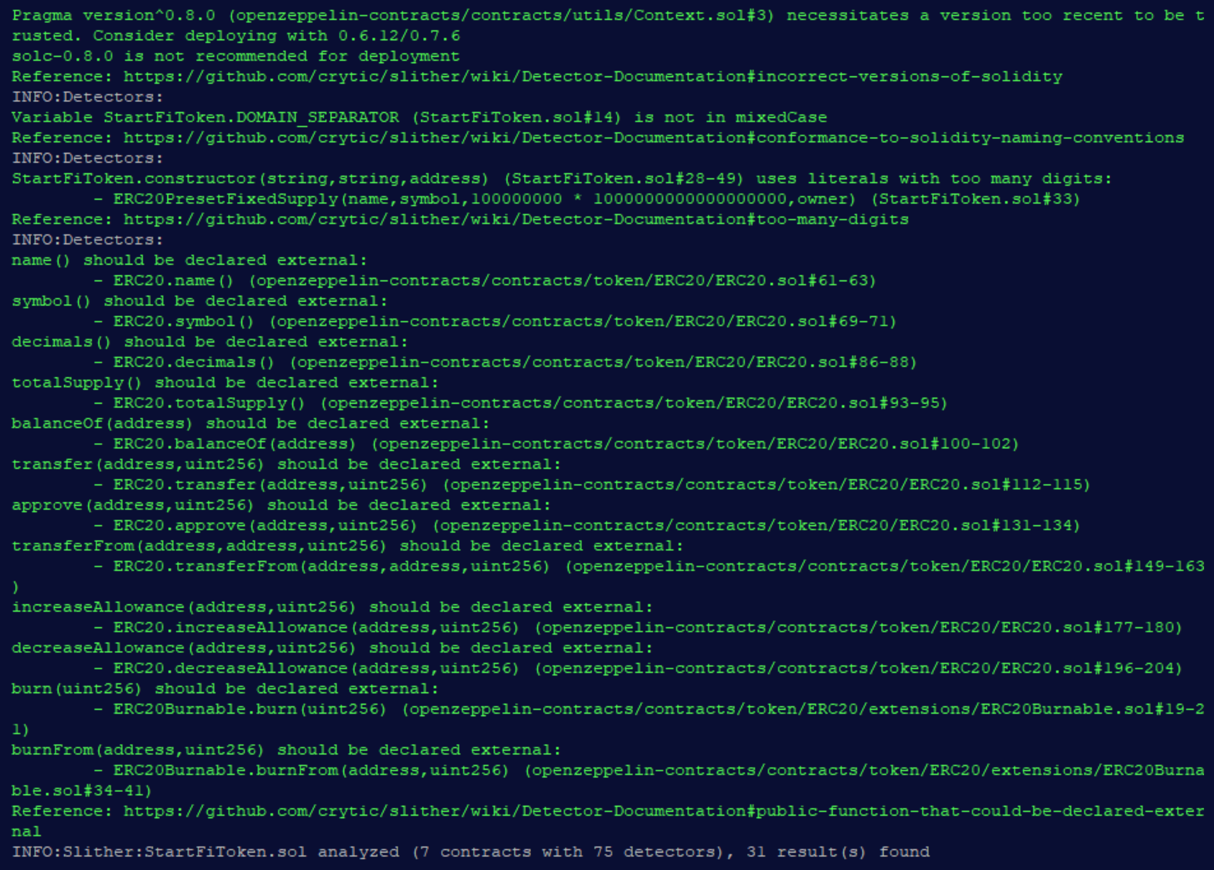
Description:

ShellBoxes augmented coverage of the specific contract areas through the use of automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:



```
INFO:Detectors:
StartFiToken.constructor(string,string,address).name (StartFiToken.sol#29) shadows:
        - ERC20.name() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#61-63) (function)
        - IERC20Metadata.name() (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#16) (func
tion)
StartFiToken.constructor(string,string,address).symbol (StartFiToken.sol#30) shadows:
        - ERC20.symbol() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#69-71) (function)
        - IERC20Metadata.symbol() (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#21) (fu
nction)
ERC20PresetFixedSupply.constructor(string,string,uint256,address).name (openzeppelin-contracts/contracts/token/ERC20/
presets/ERC20PresetFixedSupply.sol#25) shadows:
        - ERC20.name() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#61-63) (function)
        - IERC20Metadata.name() (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#16) (func
tion)
ERC20PresetFixedSupply.constructor(string,string,uint256,address).symbol (openzeppelin-contracts/contracts/token/ERC2
0/presets/ERC20PresetFixedSupply.sol#26) shadows:
        - ERC20.symbol() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#69-71) (function)
        - IERC20Metadata.symbol() (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#21) (fu
nction)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
StartFiToken.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (StartFiToken.sol#60-87) uses timestamp fo
r comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,AnyswapV3ERC20: Expired permit) (StartFiToken.sol#69)
StartFiToken.transferWithPermit(address,address,uint256,uint256,uint8,bytes32,bytes32) (StartFiToken.sol#98-129) uses
 timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,StartFiToken: Expired permit) (StartFiToken.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
StartFiToken.constructor(string,string,address) (StartFiToken.sol#28-49) uses assembly
        - INLINE ASM (StartFiToken.sol#35-37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['>=0.8.0', '^0.8.0']
        - >=0.8.0 (StartFiToken.sol#3)
        - SMTChecker (StartFiToken.sol#4)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Burnable.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/presets/ERC20PresetFixedSupply.sol#2)
        - ^0.8.0 (openzeppelin-contracts/contracts/utils/Context.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Context._msgData() (openzeppelin-contracts/contracts/utils/Context.sol#20-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.8.0 (StartFiToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0
.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to
 be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Burnable.sol#3) necessitates a ver
sion too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#3) necessitates a ve
rsion too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/presets/ERC20PresetFixedSupply.sol#2) necessitates
 a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
```



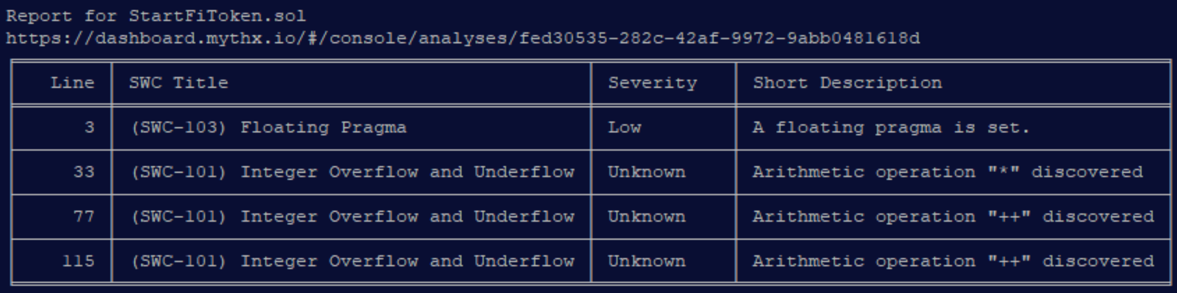
```
Pragma version^0.8.0 (openzeppelin-contracts/contracts/utils/Context.sol#3) necessitates a version too recent to be t
rusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable StartFiToken.DOMAIN_SEPARATOR (StartFiToken.sol#14) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
StartFiToken.constructor(string,string,address) (StartFiToken.sol#28-49) uses literals with too many digits:
        - ERC20PresetFixedSupply(name,symbol,100000000 * 1000000000000000000,owner) (StartFiToken.sol#33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#61-63)
symbol() should be declared external:
        - ERC20.symbol() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#69-71)
decimals() should be declared external:
        - ERC20.decimals() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#86-88)
totalSupply() should be declared external:
        - ERC20.totalSupply() (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#93-95)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#100-102)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#112-115)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#149-163
)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#196-204)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Burnable.sol#19-2
1)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Burna
ble.sol#34-41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-exter
nal
INFO:Slither:StartFiToken.sol analyzed (7 contracts with 75 detectors), 31 result(s) found
```

3.7. Automated Security Scan:

Description:

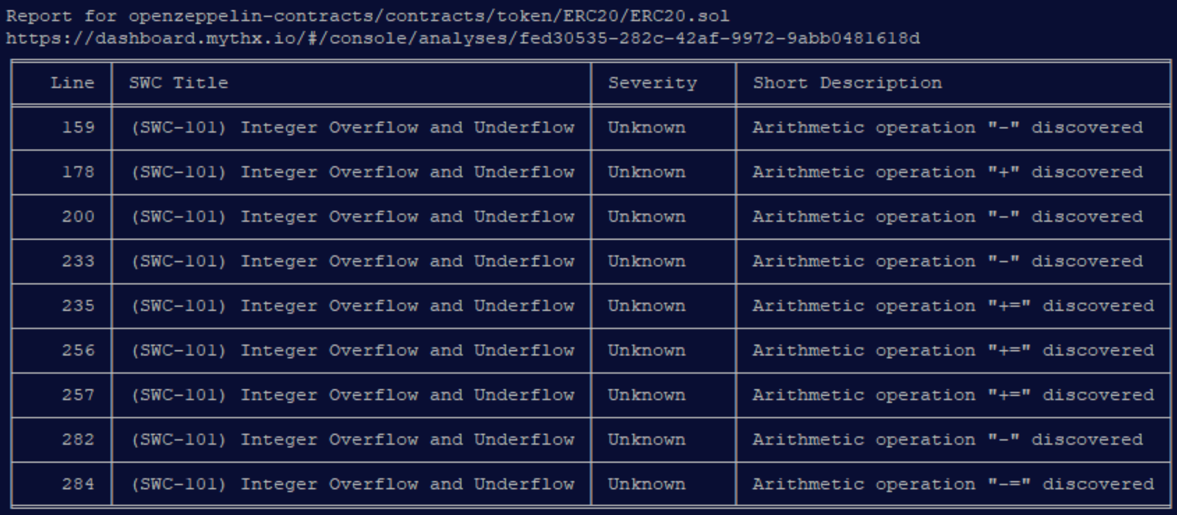
ShellBoxes used automated security scanners to assist with detection of well-known security issues and to identify vulnerabilities on the smart contract we used MythX. MythX is a security analysis API that allows anyone to create purpose-built security tools for smart contract developers.

Result:



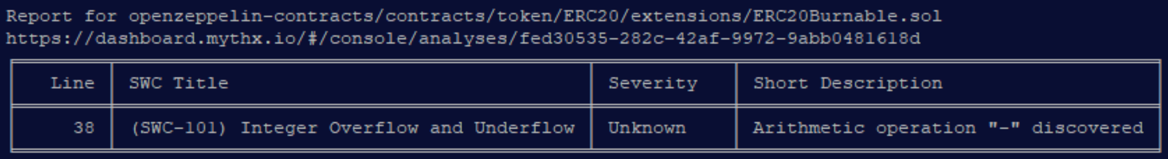
Report for StartFiToken.sol
https://dashboard.mythx.io/#/console/analyses/fed30535-282c-42af-9972-9abb0481618d

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
33	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
77	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
115	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered



Report for openzeppelin-contracts/contracts/token/ERC20/ERC20.sol
https://dashboard.mythx.io/#/console/analyses/fed30535-282c-42af-9972-9abb0481618d

Line	SWC Title	Severity	Short Description
159	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
178	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
200	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
233	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
235	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
256	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
257	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
282	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
284	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-=" discovered



Report for openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Burnable.sol
https://dashboard.mythx.io/#/console/analyses/fed30535-282c-42af-9972-9abb0481618d

Line	SWC Title	Severity	Short Description
38	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

Conclusion:

The majority of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

4. Conclusion

We examined the design and implementation of StartFI Token in this audit. The present code base is well-organized. We would much appreciate any constructive input or ideas regarding our methodology, audit findings, or potential scope/coverage gaps in this report.

SHELLBOXES

For a Contract Audit contact us at contact@shellboxes.com