VXL DOLLAR

Smart Contract Security Audit

Prepared by ShellBoxes

May 24th, 2022 - June 17th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	VXL DOLLAR
Version	1.0
Classification	Public

Scope

The VXL DOLLAR Contract in the VXL DOLLAR Repository

Repo	Commit Hash
https://github.com/vxldollar/vxldollar-tokens	744710ab1554ba69fa5c007aa539f2e7d7eaa1f4
https://github.com/vxldollar/vxldollar-tokens	4683076066a6de32be77a2ce7ed36d18133eff8c

Files	MD5 Hash
vxld-ethereum.sol	6160e6c3ad72ae676fdadaec1411b3b5
vxld-polygon.sol	5496d6b4ab50d37f22daeb2e629be5ab
vxld-tron.sol	d1c22c67a8a9029a19849c746b6602e3

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

VXL DOLLAR engaged ShellBoxes to conduct a security assessment on the VXL DOLLAR beginning on May 24th, 2022 and ending June 17th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

In response to the initial report, the VXL DOLLAR team addressed all the issues raised and put in place the necessary corrections and updates. The present code base is well-structured and ready for the mainnet.

1.1 About VXL DOLLAR

Issuer	VXL DOLLAR
Website	https://www.vxldollar.com/
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within 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 code that does not comply with 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

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the VXL DOLLAR 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 critical-severity, 1 high-severity, 2 medium-severity, 6 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Unlimited Mint Can Cause Inflation Of The Stable Coin	CRITICAL	Fixed
The UpgradedStandardToken Can Be Missused By The Owner	HIGH	Fixed
Centralization Risk Can Lead To Burning Any Tokens	MEDIUM	Fixed
Fees can be bypassed	MEDIUM	Fixed
Missing Value Verification	LOW	Fixed
Missing Address Verification	LOW	Fixed
Approve Race	LOW	Fixed
Floating Pragma	LOW	Fixed
Approve Race	LOW	Fixed
Floating Pragma	LOW	Fixed

3 Finding Details

A vxld-polygon.sol/vxld-tron.sol

A.1 Unlimited Mint Can Cause Inflation Of The Stable Coin [CRITICAL]

Description:

The VXL DOLLAR is a cryptographic token issued by VXL Dollar SA, pegged strictly 1:1 to the US dollar. The issue function offers unlimited mint to the owner which can lead to price inflation.

Code:

Listing 1: vxld-polygon.sol

```
406 function issue(uint amount) public onlyOwner {
407     require(_totalSupply + amount > _totalSupply);
408     require(balances[owner] + amount > balances[owner]);
409
410     balances[owner] += amount;
411     _totalSupply += amount;
412     Issue(amount);
413 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

It is recommended to add some restrictions concerning the mint to ensure the price stability of the token.

Status - Fixed

The VXL Dollar team has fixed the issue by removing the mint functionality.

A.2 The UpgradedStandardToken Can Be Missused By The Owner [HIGH]

Description:

The owner have the ability to deprecate this contract and delegate all the calls to a new contract by calling the deprecate function, this new contract is a black box Therefore, the owner can implement a new logic that could harm the users, such as incrementing the fees or taking their tokens.

Code:

Listing 2: vxld-polygon.sol

```
387 function deprecate(address _upgradedAddress) public onlyOwner {
388     deprecated = true;
389     upgradedAddress = _upgradedAddress;
390     Deprecate(_upgradedAddress);
391 }
```

Recommendation:

It is recommended to use a proxy contract to implement the upgradability of the token. In addition to that, consider using a multi-sig wallet as the owner's wallet to have an additional layer of security.

Status - Fixed

The VXL Dollar team has fixed the issue by removing the deprecation option.

A.3 Centralization Risk Can Lead To Burning Any Tokens [MEDIUM]

Description:

The destroyBlackFunds function gives the owner the ability to blacklist anyone and burn his tokens. This represents a centralization risk, due to the amount of power the owner has over the contract.

Code:

Listing 3: vxld-polygon.sol

```
291 function destroyBlackFunds (address _blackListedUser) public onlyOwner {
292     require(isBlackListed[_blackListedUser]);
293     uint dirtyFunds = balanceOf(_blackListedUser);
294     balances[_blackListedUser] = 0;
295     _totalSupply -= dirtyFunds;
296     DestroyedBlackFunds(_blackListedUser, dirtyFunds);
297 }
```

Recommendation:

It is recommended to either restrict the burn functionality to the user himself. If this is the behavior is intended, consider documenting it to notify the community.

Status - Fixed

The VXL Dollar team has fixed the issue by removing the burn functionality to avoid centralization risks.

A.4 Fees can be bypassed [MEDIUM]

Description:

When a user wants to transfer tokens, the contracts takes a portion of the transferred amount as fee and transfers it to the owner. The fees can be bypassed if the user sends an amount that is lower than 10000/basisPointsRate due to a type coversion issue.

Code:

Listing 4: vxld-polygon.sol

```
127     function transfer(address _to, uint _value) public
         onlyPayloadSize(2 * 32) {
128         uint fee = (_value.mul(basisPointsRate)).div(10000);
129         if (fee > maximumFee) {
130             fee = maximumFee;
131         }
132         uint sendAmount = _value.sub(fee);
133         balances[msg.sender] = balances[msg.sender].sub(_value);
134         balances[_to] = balances[_to].add(sendAmount);
135         if (fee > 0) {
136             balances[owner] = balances[owner].add(fee);
137             Transfer(msg.sender, owner, fee);
138         }
139         Transfer(msg.sender, _to, sendAmount);
140     }
```

Listing 5: vxld-polygon.sol

```
177 function transferFrom(address _from, address _to, uint _value) public
        onlyPayloadSize(2 * 32) {
178     var _allowance = allowed[_from][msg.sender];

180     // Check is not needed because sub(_allowance, _value) will already
        // throw if this condition is not met
181     // if (_value > _allowance) throw;
```

```
183     uint fee = (_value.mul(basisPointsRate)).div(10000);
184      if (fee > maximumFee) {
185         fee = maximumFee;
186     }
187     if (_allowance < MAX_UINT) {
188         allowed[_from][msg.sender] = _allowance.sub(_value);
189     }
190     uint sendAmount = _value.sub(fee);
191     balances[_from] = balances[_from].sub(_value);
192     balances[_to] = balances[_to].add(sendAmount);
193     if (fee > 0) {
194         balances[owner] = balances[owner].add(fee);
195         Transfer(_from, owner, fee);
196     }
197     Transfer(_from, _to, sendAmount);
198 }
```

Recommendation:

It is recommended to require the transferred amount to be higher than 10000/basisPointsRate.

Status - Fixed

The VXL Dollar team has fixed the issue by removing the fees from the transfer functionality.

A.5 Missing Value Verification [LOW]

Description:

The VXLDollar function lack a safety check in the values of the arguments, these values should be verified to allow only the ones that go with the contract's logic. The _initialSupply and _decimals arguments should be verified to be different from 0.

Code:

Listing 6: vxld-polygon.sol

```
330 function VXLDollar(uint _initialSupply, string _name, string _symbol,
      uint _decimals) public {
331     _totalSupply = _initialSupply;
332     name = _name;
333     symbol = _symbol;
334     decimals = _decimals;
335     balances[owner] = _initialSupply;
336     deprecated = false;
337 }
```

Recommendation:

Consider verifying the following values: `_initialSupply`and `_decimals` to be different than 0.

Status - Fixed

The VXL Dollar team has fixed the issue by removing the arguments and hard-coding the values in the constructor.

A.6 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible. The `_upgradedAddress` argument should be different from the `address(0)`.

Code:

Listing 7: vxld-polygon.sol

```
387 function deprecate(address _upgradedAddress) public onlyOwner {
388     deprecated = true;
389     upgradedAddress = _upgradedAddress;
390     Deprecate(_upgradedAddress);
391 }
```

Recommendation:

It is recommended to verify that the addresses provided in the arguments are different from the address(0).

Status - Fixed

The VXL Dollar team has fixed the issue by removing the deprecate function, knowing that the contract does not support deprecation as mentioned in the issue A2.

A.7 Approve Race [LOW]

Description:

The contract contains a widely known racing condition in its approve function, wherein a spender can 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 8: vxld-polygon.sol

```
369 function approve(address _spender, uint _value) public
        onlyPayloadSize(2 * 32) {
370     if (deprecated) {
371         return UpgradedStandardToken(upgradedAddress).approveByLegacy(
            msg.sender, _spender, _value);
372     } else {
373         return super.approve(_spender, _value);
374     }
375 }
```

Recommendation:

Use increaseAllowance and decreaseAllowance functions implemented in the ERC20 standard to modify the approval amount instead of using the approve function to modify it.

Status - Fixed

The VXL team has fixed the issue by implementing the use of the increaseAllowance and decreaseAllowance functions.

A.8 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.4.17. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 9: vxld-polygon.sol

```
5 pragma solidity ^0.4.17;
```

Recommendation:

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

Status - Fixed

The VXL Dollar team has fixed the issue by locking the pragma version to 0.8.14.

B vxld-ethereum.sol

B.1 Approve Race [LOW]

Description:

The contract contains a widely known racing condition in its approve function, wherein a spender can witness the token owner broadcast a transaction altering their approval and quickly sigingn 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 10: vxld-ethereum.sol

```
139  function approve(address spender, uint tokens) public returns (bool
140 success) {
141  allowed[msg.sender][spender] = tokens;
```

```
142   emit Approval(msg.sender, spender, tokens);
143   return true;
144   }
```

Recommendation:

Use increaseAllowance and decreaseAllowance functions implemented in the ERC20 standard to modify the approval amount instead of using the approve function to modify it.

Status - Fixed

The VXL team has fixed the issue by implementing the use of the increaseAllowance and decreaseAllowance functions.

B.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.4.24. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 11: vxld-ethereum.sol

```
1  pragma solidity ^0.4.24;
```

Recommendation:

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

Status - Fixed

The VXL Dollar team has fixed the issue by locking the pragma version to 0.8.14.

4 Best Practices

BP.1 Update The Pragma Version

Description:

The contract uses the solidity version 0.4.17, this version is quite old as solidity have the 0.8.x version. Consider updating the solidity version to one of the latest ones to make use of the new functionalities and avoid the old versions bugs.

Code:

Listing 12: vxld-polygon.sol

```
5 pragma solidity ^0.4.17;
```

BP.2 Unnecessary Initializations

Description:

When a variable is declared in solidity, it gets initialized with its type's default value. Thus, there is no need to initialize a variable with the default value.

Code:

Listing 13: vxld-polygon.sol

```
109 // additional variables for use if transaction fees ever became necessary
110 uint public basisPointsRate = 0;
111 uint public maximumFee = 0;
```

BP.3 Update The Pragma Version

Description:

The contract uses the solidity version 0.4.24, this version is quite old as solidity have the 0.8.x version. Consider updating the solidity version to one of the latest ones to make use of the new functionalities and avoid the old versions bugs.

Code:

Listing 14: vxld-ethereum.sol

```
1 pragma solidity ^0.4.24;
```

5 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using 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:

```
Contract locking ether found:
        Contract VXL (vxld-ethereum.sol#75-197) has payable functions:
         - VXL.fallback() (vxld-ethereum.sol#194-196)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#co
ntracts-that-lock-ether

Pragma version^0.4.24 (vxld-ethereum.sol#1) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#in
correct-versions-of-solidity

Variable VXL._totalSupply (vxld-ethereum.sol#79) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#co
nformance-to-solidity-naming-conventions

VXL.constructor() (vxld-ethereum.sol#87-96) uses literals with too many dig
its:
        - _totalSupply = 20000000000000000000 (vxld-ethereum.sol#91)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#to
o-many-digits
```

```
safeMul(uint256,uint256) should be declared external:
	- SafeMath.safeMul(uint256,uint256) (vxld-ethereum.sol#35-38)
safeDiv(uint256,uint256) should be declared external:
	- SafeMath.safeDiv(uint256,uint256) (vxld-ethereum.sol#39-42)
totalSupply() should be declared external:
	- ERC20Interface.totalSupply() (vxld-ethereum.sol#49)
	- VXL.totalSupply() (vxld-ethereum.sol#102-104)
balanceOf(address) should be declared external:
	- ERC20Interface.balanceOf(address) (vxld-ethereum.sol#50-51)
	- VXL.balanceOf(address) (vxld-ethereum.sol#110-113)
allowance(address,address) should be declared external:
	- ERC20Interface.allowance(address,address) (vxld-ethereum.sol#52-5
3)
	- VXL.allowance(address,address) (vxld-ethereum.sol#170-173)
transfer(address,uint256) should be declared external:
	- ERC20Interface.transfer(address,uint256) (vxld-ethereum.sol#54)
	- VXL.transfer(address,uint256) (vxld-ethereum.sol#121-127)
approve(address,uint256) should be declared external:
	- ERC20Interface.approve(address,uint256) (vxld-ethereum.sol#55-56)
	- VXL.approve(address,uint256) (vxld-ethereum.sol#138-143)
transferFrom(address,address,uint256) should be declared external:
	- ERC20Interface.transferFrom(address,address,uint256) (vxld-ethere
um.sol#57-58)
	- VXL.transferFrom(address,address,uint256) (vxld-ethereum.sol#155-
163)
approveAndCall(address,uint256,bytes) should be declared external:
	- VXL.approveAndCall(address,uint256,bytes) (vxld-ethereum.sol#181-
188)
fallback() should be declared external:
	- VXL.fallback() (vxld-ethereum.sol#194-196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pu
blic-function-that-could-be-declared-external
```

```
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:StandardToken.transferFrom(address,address,uint256) (vxld-poly
gon.sol#171-192)
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:StandardToken.approve(address,uint256) (vxld-polygon.sol#199-2
09)
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:ERC20.transferFrom(address,address,uint256) (vxld-polygon.sol#
95)
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:ERC20.approve(address,uint256) (vxld-polygon.sol#96)
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:ERC20Basic.transfer(address,uint256) (vxld-polygon.sol#85)
UpgradedStandardToken (vxld-polygon.sol#307-313) has incorrect ERC20 functi
on interface:BasicToken.transfer(address,uint256) (vxld-polygon.sol#126-139
)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:BasicToken.transfer(address,uint256) (vxld-polygon.sol#126-139)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:ERC20Basic.transfer(address,uint256) (vxld-polygon.sol#85)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:StandardToken.transferFrom(address,address,uint256) (vxld-polygon.sol#171-
192)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:StandardToken.approve(address,uint256) (vxld-polygon.sol#199-209)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:ERC20.transferFrom(address,address,uint256) (vxld-polygon.sol#95)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:ERC20.approve(address,uint256) (vxld-polygon.sol#96)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:VXLDollar.transfer(address,uint256) (vxld-polygon.sol#340-347)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
:VXLDollar.transferFrom(address,address,uint256) (vxld-polygon.sol#350-357)
VXLDollar (vxld-polygon.sol#315-451) has incorrect ERC20 function interface
```

```
:VXLDollar.approve(address,uint256) (vxld-polygon.sol#369-375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#in
correct-erc20-interface

Ownable.transferOwnership(address) (vxld-polygon.sol#68-72) should emit an
event for:
        - owner = newOwner (vxld-polygon.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#mi
ssing-events-access-control

VXLDollar.deprecate(address)._upgradedAddress (vxld-polygon.sol#387) lacks
a zero-check on :
                - upgradedAddress = _upgradedAddress (vxld-polygon.sol#389)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#mi
ssing-zero-address-validation

Pragma version^0.4.17 (vxld-polygon.sol#5) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#in
correct-versions-of-solidity

Variable ERC20Basic._totalSupply (vxld-polygon.sol#82) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._to (vxld-polygon.sol#126) i
s not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (vxld-polygon.sol#126
) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (vxld-polygon.sol#146) is no
t in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (vxld-p
olygon.sol#171) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (vxld-pol
ygon.sol#171) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (vxld-
polygon.sol#171) is not in mixedCase
```

```
Parameter StandardToken.approve(address,uint256)._spender (vxld-polygon.sol
#199) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (vxld-polygon.sol#1
99) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (vxld-polygon.sol
#217) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (vxld-polygon.s
ol#217) is not in mixedCase
Parameter BlackList.getBlackListStatus(address)._maker (vxld-polygon.sol#27
1) is not in mixedCase
Parameter BlackList.addBlackList(address)._evilUser (vxld-polygon.sol#281)
is not in mixedCase
Parameter BlackList.removeBlackList(address)._clearedUser (vxld-polygon.sol
#286) is not in mixedCase
Parameter BlackList.destroyBlackFunds(address)._blackListedUser (vxld-polyg
on.sol#291) is not in mixedCase
Parameter VXLDollar.transfer(address,uint256)._to (vxld-polygon.sol#340) is
 not in mixedCase
Parameter VXLDollar.transfer(address,uint256)._value (vxld-polygon.sol#340)
 is not in mixedCase
Parameter VXLDollar.transferFrom(address,address,uint256)._from (vxld-polyg
on.sol#350) is not in mixedCase
Parameter VXLDollar.transferFrom(address,address,uint256)._to (vxld-polygon
.sol#350) is not in mixedCase
Parameter VXLDollar.transferFrom(address,address,uint256)._value (vxld-poly
gon.sol#350) is not in mixedCase
Parameter VXLDollar.approve(address,uint256)._spender (vxld-polygon.sol#369
) is not in mixedCase
Parameter VXLDollar.approve(address,uint256)._value (vxld-polygon.sol#369)
is not in mixedCase
Parameter VXLDollar.allowance(address,address)._owner (vxld-polygon.sol#378
) is not in mixedCase
Parameter VXLDollar.allowance(address,address)._spender (vxld-polygon.sol#3
78) is not in mixedCase
```

```
Parameter VXLDollar.deprecate(address)._upgradedAddress (vxld-polygon.sol#3
87) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#co
nformance-to-solidity-naming-conventions

UpgradedStandardToken (vxld-polygon.sol#307-313) does not implement functio
ns:
        - UpgradedStandardToken.approveByLegacy(address,address,uint256) (v
xld-polygon.sol#312)
        - ERC20Basic.totalSupply() (vxld-polygon.sol#83)
        - UpgradedStandardToken.transferByLegacy(address,address,uint256) (
vxld-polygon.sol#310)
        - UpgradedStandardToken.transferFromByLegacy(address,address,addres
s,uint256) (vxld-polygon.sol#311)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
implemented-functions

transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (vxld-polygon.sol#68-72)
totalSupply() should be declared external:
        - ERC20Basic.totalSupply() (vxld-polygon.sol#83)
        - VXLDollar.totalSupply() (vxld-polygon.sol#394-400)
pause() should be declared external:
        - Pausable.pause() (vxld-polygon.sol#254-257)
unpause() should be declared external:
        - Pausable.unpause() (vxld-polygon.sol#262-265)
addBlackList(address) should be declared external:
        - BlackList.addBlackList(address) (vxld-polygon.sol#281-284)
removeBlackList(address) should be declared external:
        - BlackList.removeBlackList(address) (vxld-polygon.sol#286-289)
destroyBlackFunds(address) should be declared external:
        - BlackList.destroyBlackFunds(address) (vxld-polygon.sol#291-297)
transferByLegacy(address,address,uint256) should be declared external:
        - UpgradedStandardToken.transferByLegacy(address,address,uint256) (
```

```
vxld-polygon.sol#310)
transferFromByLegacy(address,address,address,uint256) should be declared ex
ternal:
        - UpgradedStandardToken.transferFromByLegacy(address,address,addres
s,uint256) (vxld-polygon.sol#311)
approveByLegacy(address,address,uint256) should be declared external:
        - UpgradedStandardToken.approveByLegacy(address,address,uint256) (v
xld-polygon.sol#312)
deprecate(address) should be declared external:
        - VXLDollar.deprecate(address) (vxld-polygon.sol#387-391)
issue(uint256) should be declared external:
        - VXLDollar.issue(uint256) (vxld-polygon.sol#406-413)
redeem(uint256) should be declared external:
        - VXLDollar.redeem(uint256) (vxld-polygon.sol#420-427)
setParams(uint256,uint256) should be declared external:
        - VXLDollar.setParams(uint256,uint256) (vxld-polygon.sol#429-438)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pu
blic-function-that-could-be-declared-external
. analyzed (13 contracts with 75 detectors), 74 result(s) found
```

Conclusion:

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

6 Conclusion

In this audit, we examined the design and implementation of VXL DOLLAR contract and discovered several issues of varying severity. VXL DOLLAR team addressed all the issues raised in the initial report and implemented the necessary fixes.

The present code base is well-structured and ready for the mainnet.

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