



SMART CONTRACT AUDIT

Project: Xend Finance
Date: April 26th, 2022

TABLE OF CONTENTS

Summary	03
Scope of Work	05
Workflow of the auditing process	07
Structure and organization of the findings	08
Manual Report	09
■ High Resolved	
Possible broke of contracts by upgradable dependency	09
■ High Resolved	
Possible broke of contract by transferring tokens to it	09
■ High Resolved	
Missed state update in xVault	10
■ High Resolved	
Not used <i>setProtectedToken</i> in Strategy	10
■ High Resolved	
Possible loss in xVault contract	11
■ High Resolved	
Wrong interacting with fulcrum contract in xAAVE and xBUSD	11
■ High Resolved	
Incorrect profit logic in xAAVE and xBUSD contracts	12
■ Medium Resolved	
Possibility to exceed maximum strategies number at xVault.sol	13
■ Medium Resolved	
Ignore the return value of transfer at xVault.sol and BaseStrategy.sol	13
■ Medium Resolved	
Wrong approve tokens in xAuto contracts	14



Low Resolved	
Optimization at XAAVE.sol and XUSDT.sol	14
Low Resolved	
Variables can be declared as constant at xVault.sol	15
Low Resolved	
Optimization at xVault.sol	15
Low Resolved	
Lacks a zero-check in the constructor of xVault.sol and functions	15
Low Resolved	
Illogical use of deposit limit at xVault	16
Low Resolved	
Wrong predefined apr address as XAAVE	16
Low Resolved	
Missed check for the token in EarnAPYWithPool	16
Informational Resolved	
Unnecessary event and import statement at xVault.sol	17
Informational Resolved	
Commented functions at xVault.sol	17
Informational Resolved	
Not proper use of assert at xVault.sol	17
Informational Resolved	
Not proper use of assert at XAAVE.sol and XUSDT.sol	18
Test results	19
Tests are written by Vidma	20

SUMMARY

Vidma team has conducted a smart contract audit for the given codebase.

The contracts are in good condition. Based on the fixes provided by the Xend Finance team and on the quality and security of the codebase provided, Vidma team can give a score of 99 to the audited smart contracts.

During the auditing process, the Vidma team has found a couple of informational issues, 7 issues with a low level of severity, 3 issues with a medium level of severity, and 7 issues with a high level of severity. No critical issues were supposed.

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	7 issues	7 issues	0 issues
Medium	3 issues	3 issues	0 issues
Low	7 issue	7 issues	0 issues
Informational	4 issues	4 issues	0 issues
Total	21 issues	21 issues	0 issues

Evaluating the findings, we can assure that the contract is safe to use and all the issues found are performed only by certain conditions and cases. Under the given circumstances we can set the following risk level:

High Confidence



Based on the given findings, risk level, performance, and code style, Vidma team can grant the following overall score:

Vidma auditing team has conducted a bunch of integrated autotests to ensure that the given codebase has decent performance and security levels. The test results and the coverage can be found in the accompanying section of this audit report.

Please mind that this audit does not certify the definite reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by Vidma auditing team. If the code is under development, we recommend run one more audit once the code is finalized.



SCOPE OF WORK



Credit Unions, Cooperatives, and Individuals anywhere in the world can now earn higher interests in stable currencies on their savings.

Within the scope of this audit, two independent auditors deeply investigated the given codebase and analyzed the overall security and performance of smart contracts.

Vidma auditing team has made a review of the following contracts:

- APRWithPoolOracle;
- EarnAPRWithPool;
- XUSDT;
- XAAVE;
- xVault;
- BaseStrategy;
- strategyUgoHawkVenusFarm.

The source code was taken from the following **sources**:

- <https://github.com/xendfinance/polygon-earn/commit/aecc65015a227ad56995ef78cab7643eebe3d0eb>
- <https://github.com/xendfinance/x-vault/commit/188d55c68c02f84597ce35ad67ab90443ca503f3>

Initial commits submitted for the audit:

- [188d55c68c02f84597ce35ad67ab90443ca503f3](https://github.com/xendfinance/polygon-earn/commit/aecc65015a227ad56995ef78cab7643eebe3d0eb)
- [aecc65015a227ad56995ef78cab7643eebe3d0eb](https://github.com/xendfinance/polygon-earn/commit/aecc65015a227ad56995ef78cab7643eebe3d0eb)

Last commits:

- [ef77efc3042018aaa2679ad302b16912490cba41](https://github.com/xendfinance/polygon-earn/commit/ef77efc3042018aaa2679ad302b16912490cba41)
- [c53a2ce6a88554a1f68b99e3f5236e0409ef76b9](https://github.com/xendfinance/polygon-earn/commit/c53a2ce6a88554a1f68b99e3f5236e0409ef76b9)



In order to conduct a more detailed audit, Xend Finance has provided the following **documentation**:

<https://drive.google.com/drive/folders/1ZxqTao1fuq41rnr9BbMDV4wdkxWEWL9V?usp=sharing>

WORKFLOW OF THE AUDITING PROCESS

During the manual phase of the audit, Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract.

Within the testing part, Vidma auditors run integration tests using the Truffle testing framework. The test coverage and the tests themselves are inserted into this audit report.

Vidma team uses the most sophisticated and contemporary methods and techniques to ensure the contract does not have any vulnerabilities or security risks:

- Re-entrancy;
- Access Management Hierarchy;
- Arithmetic Over/Under Flows;
- Unexpected Ether;
- Delegatecall;
- Default Public Visibility;
- Hidden Malicious Code;
- Entropy Illusion (Lack of Randomness);
- External Contract Referencing;
- Short Address/Parameter Attack;
- Unchecked CALL Return Values;
- Race Conditions / Front Running;
- General Denial Of Service (DOS);
- Uninitialized Storage Pointers;
- Floating Points and Precision;
- Tx.Origin Authentication;
- Signatures Replay;
- Pool Asset Security (backdoors in the underlying ERC-20).

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For the convenience of reviewing the findings in this report, Vidma auditors classified them in accordance with the severity of the issues. (from most critical to least critical). The acceptance criteria are described below.

All issues are marked as "Resolved" or "Unresolved", depending on whether they have been fixed by Xend Finance or not. The latest commit, indicated in this audit report should include all the fixes made.

To ease the explanation, the Vidma team has provided a detailed description of the issues and recommendations on how to fix them.

Hence, according to the statements above, we classified all the findings in the following way:

Finding	Description
■ Critical	The issue bear a definite risk to the contract, so it may affect the ability to compile or operate.
■ High	Major security or operational risk found, that may harm the end-user or the overall performance of the contract.
■ Medium	The issue affects the contract to operate in a way that doesn't significantly hinder its performance.
■ Low	The found issue has a slight impact on the performance of the contract or its security.
■ Informational	The issue does not affect the performance or security of the contract/recommendations on the improvements.

MANUAL REPORT

Possible broke of contracts by upgradable dependency

 High | Resolved

Contract Strategy dependence on pre-deployed proxy contract crUSDT. It means that this proxy contract can be updated and the logic of the Strategy will be broken. In that case, xVault can not be able to withdraw funds from Strategy and return it to users.

Recommendation:

Implement proxy pattern for Strategy to be able to fix deployed the contract.

Possible broke of contract by transferring tokens to it

 High | Resolved

Contract xVault has a `tokenBalance` variable to store the number of tokens on it. It can't be updated when someone transfers by calling `token.transfer`. In this case, `balanceOf` and `tokenBalance` will return different values which can lead to miscalculations and subtraction overflow.

Recommendation:

Add update for `tokenBalance` on top of withdrawal method.

```
) public nonReentrant returns (uint256) {  
    tokenBalance = token.balanceOf(address(this));  
    uint256 shares = maxShare;
```

Missed state update in xVault

High | Resolved

Contract xVault has a *tokenBalance* variable to store the balance of tokens in the contract. It is updated whenever xVault receives or sends tokens except in cases when strategy withdrawn less than the user actually wants to receive.

```
if (value > token.balanceOf(address(this))) {  
    value = token.balanceOf(address(this));  
    shares = _sharesForAmount(value);  
}
```

In this case, the value of actually withdrawn tokens should be equal to *tokenBalance* but it's not. So compile throws runtime error while trying to do subtraction after token transfer at:

```
tokenBalance = tokenBalance.sub(value);
```

Recommendation:

Update the state of *tokenBalance* inside case mentioned above.

Not used *setProtectedToken* in Strategy

High | Resolved

Strategy contracts do not allow the withdrawal of protected tokens from contracts. To protect them from being withdrawn they should be added to the 'protected' list.

Recommendation:

Use *setProtectedToken* in BaseStrategy constructor.

Possible loss in xVault contract

High | Resolved

Contract xVault withdraws tokens from Strategy if it doesn't have enough to pay users. Withdraw from strategy calculate the loss of investment. After all withdrawals there is a check for *totalLoss* to be less than allowed. But in the case when tokens after withdrawal are not enough check for loss so not included which may lead to an unexpected loss for the user.

Recommendation:

Move check for a loss right before the actual transfer. And move the *totalLoss* variable to the top level of the function scope.

```
require(totalLoss <= maxLoss.mul(value.add(totalLoss)).div(MAX_BPS),  
"revert if totalLoss is more than permitted");  
token.safeTransfer(recipient, value);
```

Wrong interacting with fulcrum contract in xAAVE and xBUSD

High | Resolved

Contracts xAAVE and xBUSD deposit their funds to fulcrum contract. It did not happen as expected because the call failed with the error SafeERC20: low-level call failed.

Recommendation:

Recheck the logic behind interacting xAAVE and xBUSD with fulcrum contracts.

```
require(totalLoss <= maxLoss.mul(value.add(totalLoss)).div(MAX_BPS),  
"revert if totalLoss is more than permitted");  
token.safeTransfer(recipient, value);
```

Incorrect profit logic in xAAVE and xBUSD contracts

High | Resolved

Contracts xAAVE and xBUSD interact with different contracts to deposit users' funds in the best place and generate profit. It means that the contract should increase the pool of itself to allow users to withdraw more than they deposited initially. But it's not happening.

Recommendation:

Recheck logic of profit calculation and interact with aave, fulcrum, and fortube contracts.

Re-Audit:

In xUSDT contract in some cases 'pool' can be less than *totalDepositedAmount* which can lead to subtraction overflow. Test cases:

```
When Default
  ✓ Should deposit [0] (4448ms)
  ✓ Should deposit [1] (3283ms)
pool 3263
balanceFulcrum 0
balanceAave 3263
  ✓ Should withdraw [0] (3711ms)
  ✓ Should withdraw [1] (4058ms)
Owner
  ✓ Should change lender (1044ms)
  ✓ Should set new fee (265ms)
  ✓ Should set new fee address (241ms)
  ✓ Should set new fee precision
When FULCRUM
  ✓ Should deposit [0] (3922ms)
  ✓ Should deposit [1] (3059ms)
pool 2998
balanceFulcrum 2983
balanceAave 0
```

Deposited values 1000 and 2000, but pool value is only 2998.

Recommendation 2:

Add check if pool is bigger than `totalDepositedAmount`.

```
uint256 fee = pool > totalDepositedAmount  
?  
pool.sub(totalDepositedAmount).mul(feeAmount).div(feePrecision)  
: 0;
```

Possibility to exceed maximum strategies number at xVault.sol

Medium | Resolved

Function `setWithdrawalQueue()` and `addStrategy()` allows to exceed a `MAXIMUM_STRATEGIES` number.

Recommendation:

Add check for `withdrawalQueue` length for those functions.

Ignore the return value of transfer at xVault.sol and BaseStrategy.sol

Medium | Resolved

In function `report()` at xVault.sol `tokenBalance` is updating but it ignores return value by `token.transfer()` and `token.transferFrom()`.

In function `distributeRewards()` at BaseStrategy no check provided for return value of transfer.

Recommendation:

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Wrong approve tokens in xAuto contracts

Medium | Resolved

In xAuto contracts there are different lenders. xAuto approves tokens for active lenders on initialize function. When the owner activates a not active initial lender contract calls rebalance function to deposit on a new lender but since this lender tokens are not approved it can't transfer tokens from xAuto.

Recommendation:

One of few:

- 1) Approve all lenders on initialization;
- 2) Approve a new lender before rebalancing in `activateLender` function.

Re-Audit:

Fixed.

Optimization at XAAVE.sol and XUSDT.sol

Low | Resolved

Functions `balanceFortubeInToken()` at XUSDT.sol, `balanceFulcrumInToken()` at XAAVE.sol and XUSDT.sol are fully duplicates of internal functions `_balanceFortubeInToken()`, `_balanceFulcrumInToken()`.

Function `_rebalance()` at XAAVE.sol and XUSDT.sol is internal and never used.

Recommendation:

In public functions `balanceFortubeInToken()` and `balanceFulcrumInToken()` you can call their represented internal functions and return it's value. If `_rebalance()` is unnecessary then just remove it.

Variables can be declared as constant at xVault.sol

Low | Resolved

`MAX_BPS` and `SECS_PER_YEAR` can be declared as constant.

Recommendation:

Change those variables to constant.

Optimization at xVault.sol

Low | Resolved

Multiple usage of require statements

```
msg.sender == governance, msg.sender == management,  
msg.sender == guardian.
```

Recommendation:

Create a modifier for those checks or add `AccessControl` from OpenZeppelin where you can set all needed roles for access to specific functions.

Lacks a zero-check in the constructor of xVault.sol and functions

Low | Resolved

Constructor parameter `_governance` and functions `setTreasury()`, `setGuardian()`, `setGovernance()` have no check for zero address.

Recommendation:

Check that the address is not zero.

Illogical use of deposit limit at xVault

 Low | Resolved

Contract xVault in `deposit` function has a check for input amount of zero to deposit all possible allowed funds. It is calculated by `depositLimit - totalAssets` which can lead to subtraction underflow. This check is not included if the amount is greater than zero, which means that users can't deposit more than the limit when they deposit zero but can when more.

Recommendation:

If the logic of checking only when the amount is zero is correct then need to add check for underflow or even leave it as it is. Otherwise need to add check the global scope of function.

Wrong predefined apr address as XAAVE

 Low | Resolved

Apr address at the XAAVE contract in a polygon is not a contract.

Recommendation:

Put correct address.

Missed check for the token in EarnAPYWithPool

 Low | Resolved

Contract EarnAPYWithPool has 2 mappings of tokens in fulcrum and fortube to check if the token exists on this platform. For aave this check is missed which means that some tokens cannot be processed in `getAPROptionsInc`.

Recommendation:

Add check for existing token on aave platform.

Unnecessary event and import statement at xVault.sol

 Informational | Resolved

All related functionality to event `UpdateGuestList` is commented and not used farther in the code.

IERC20 is not used.

Recommendation:

Delete this event and all related commented functionality, delete unused import statement.

Commented functions at xVault.sol

 Informational | Resolved

Functions `setName()` and `setSymbol()` are commented.

Recommendation:

Remove commented functionality.

Not proper use of assert at xVault.sol

 Informational | Resolved

The assert function should only be used to test for internal errors and to check invariants.

Recommendation:

Require function should be used to ensure valid conditions. Replace assert with require with corresponding revert message.

Not proper use of assert at XAAVE.sol and XUSDT.sol

 Informational | Resolved

In function `set_new_feePrecision()` assert is used to check parameters of `_newFeePrecision`.

The assert function should only be used to test for internal errors and to check invariants.

Recommendation:

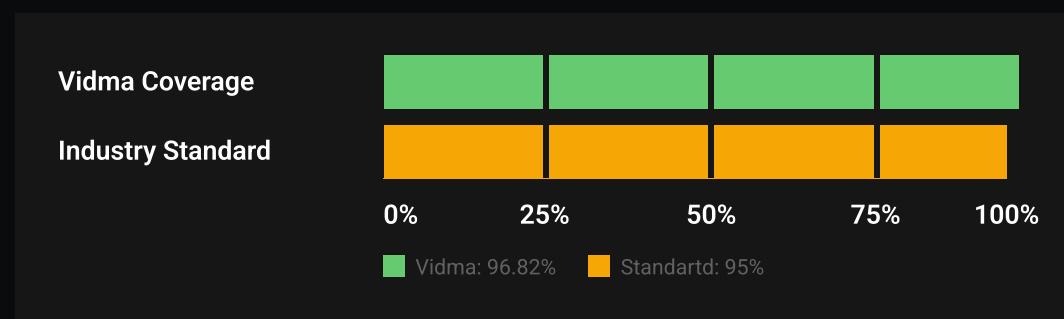
Require function should be used to ensure valid conditions. Replace assert with require with corresponding revert message.

TEST RESULTS

To verify the contract security and performance a bunch of integration tests was made using the Truffle testing framework.

Tests were based on the functionality of the code, business logic, and requirements and for the purpose of finding the vulnerabilities in the contracts.

In this section, we provide tests written by Vidma auditors.



It's important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Xend Finance repo. We write totally separate tests with code coverage of a minimum of 95%, to meet the industry standards.

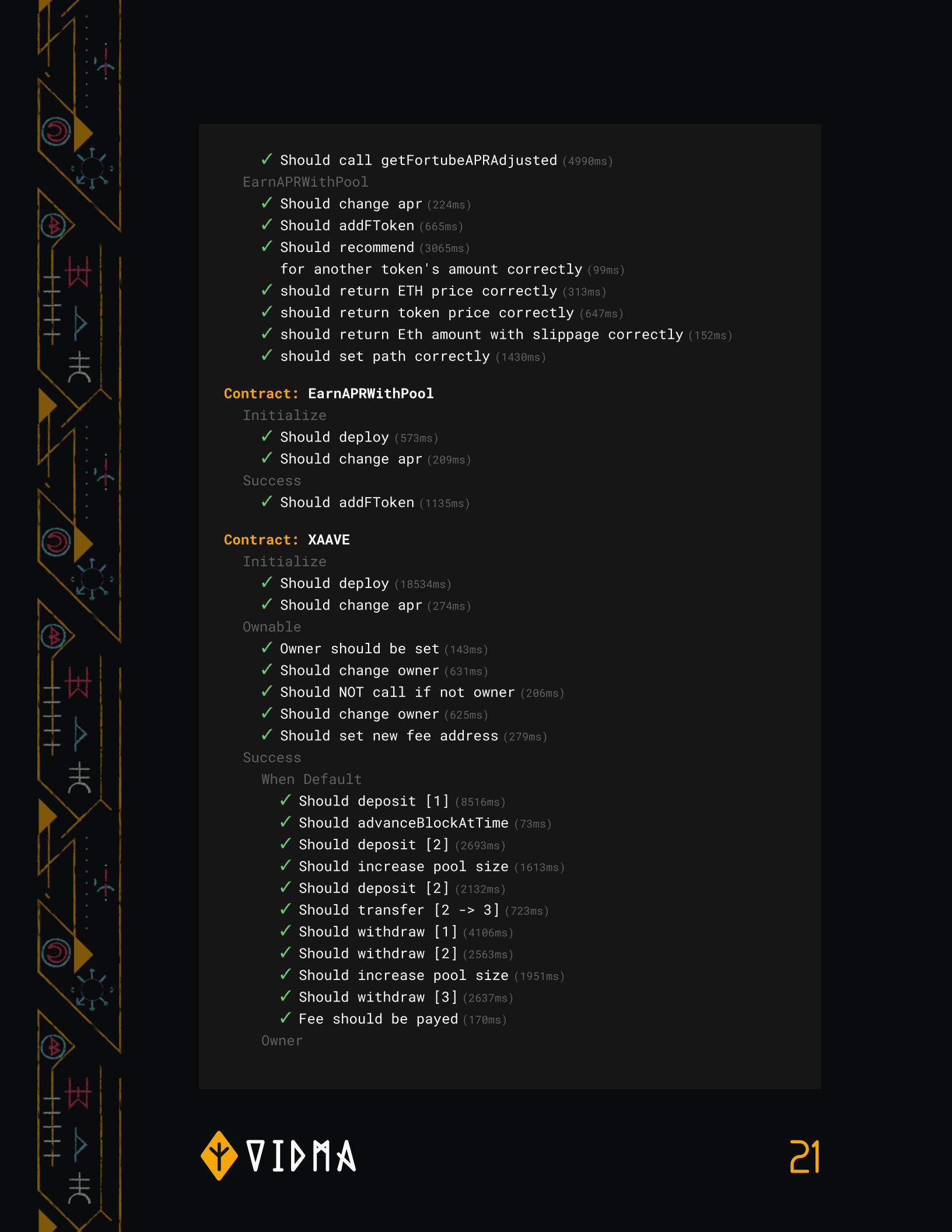
Tests are written by Vidma

Test Coverage

File	%Stmts	%Branch	%Funcs	%Lines
APRWithPoolOracle	100.00	100.00	100.00	100.00
EarnAPRWithPool	100.00	80.00	100.00	100.00
XAAVE	96.92	67.78	98.18	96.91
XAAVEProxy	100	100	0.00	100.00
XUSDT	97.30	74.49	95.16	97.26
xVault	97.66	64.47	100.00	96.92
BaseStrategy	96.47	61.90	100.00	97.62
Strategy	91.46	69.72	97.22	91.13
All files	96.52	76.24	87.65	96.88

Test Results

```
Contract: EarnAPRWithPool/APRWithPoolOracle
Initialize
  ✓ Should deploy (3003ms)
APRWithPoolOracle
  ✓ Should call getFulcrumAPRAdjusted (5714ms)
  ✓ Should call getAaveAPRAdjusted (8278ms)
```



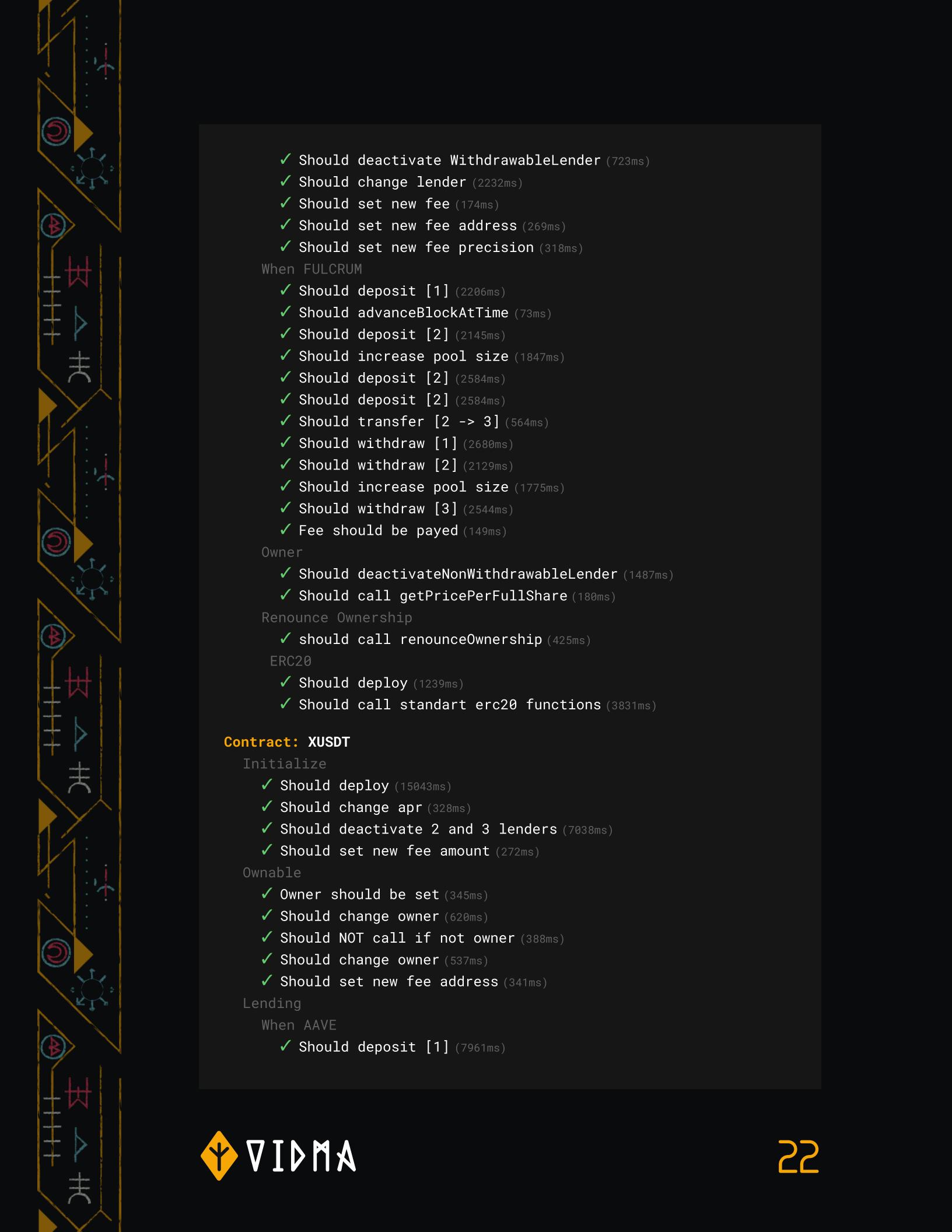
```
✓ Should call getFortubeAPRAdjusted (4990ms)
EarnAPRWithPool
✓ Should change apr (224ms)
✓ Should addFToken (665ms)
✓ Should recommend (3065ms)
  for another token's amount correctly (99ms)
✓ should return ETH price correctly (313ms)
✓ should return token price correctly (647ms)
✓ should return Eth amount with slippage correctly (152ms)
✓ should set path correctly (1430ms)
```

Contract: EarnAPRWithPool

```
Initialize
✓ Should deploy (573ms)
✓ Should change apr (209ms)
Success
✓ Should addFToken (1135ms)
```

Contract: XAAVE

```
Initialize
✓ Should deploy (18534ms)
✓ Should change apr (274ms)
Ownable
✓ Owner should be set (143ms)
✓ Should change owner (631ms)
✓ Should NOT call if not owner (206ms)
✓ Should change owner (625ms)
✓ Should set new fee address (279ms)
Success
  When Default
✓ Should deposit [1] (8516ms)
✓ Should advanceBlockAtTime (73ms)
✓ Should deposit [2] (2693ms)
✓ Should increase pool size (1613ms)
✓ Should deposit [2] (2132ms)
✓ Should transfer [2 -> 3] (723ms)
✓ Should withdraw [1] (4106ms)
✓ Should withdraw [2] (2563ms)
✓ Should increase pool size (1951ms)
✓ Should withdraw [3] (2637ms)
✓ Fee should be payed (170ms)
Owner
```



```

✓ Should deactivate WithdrawableLender (723ms)
✓ Should change lender (2232ms)
✓ Should set new fee (174ms)
✓ Should set new fee address (269ms)
✓ Should set new fee precision (318ms)

When FULCRUM
✓ Should deposit [1] (2206ms)
✓ Should advanceBlockAtTime (73ms)
✓ Should deposit [2] (2145ms)
✓ Should increase pool size (1847ms)
✓ Should deposit [2] (2584ms)
✓ Should deposit [2] (2584ms)
✓ Should transfer [2 -> 3] (564ms)
✓ Should withdraw [1] (2680ms)
✓ Should withdraw [2] (2129ms)
✓ Should increase pool size (1775ms)
✓ Should withdraw [3] (2544ms)
✓ Fee should be payed (149ms)

Owner
✓ Should deactivateNonWithdrawableLender (1487ms)
✓ Should call getPricePerFullShare (180ms)

Renounce Ownership
✓ should call renounceOwnership (425ms)

ERC20
✓ Should deploy (1239ms)
✓ Should call standart erc20 functions (3831ms)

```

Contract: XUSDT

```

Initialize
✓ Should deploy (15043ms)
✓ Should change apr (328ms)
✓ Should deactivate 2 and 3 lenders (7038ms)
✓ Should set new fee amount (272ms)

Ownable
✓ Owner should be set (345ms)
✓ Should change owner (620ms)
✓ Should NOT call if not owner (388ms)
✓ Should change owner (537ms)
✓ Should set new fee address (341ms)

Lending
When AAVE
✓ Should deposit [1] (7961ms)

```

- 
- 
- ✓ Should advanceBlockAtTime (156ms)
 - ✓ Should deposit [2] (4744ms)
 - ✓ Should increase pool size (2598ms)
 - ✓ Should deposit [2] (3925ms)
 - ✓ Should transfer [2 -> 3] (937ms)
 - ✓ Should withdraw [1] (3570ms)
 - ✓ Should withdraw [2] (3039ms)
 - ✓ Should increase pool size (2595ms)
 - ✓ Should withdraw [3] (4808ms)
 - ✓ Fee should be payed (199ms)

Owner

- ✓ Should deactivate WithdrawableLender (1588ms)
- ✓ Should change lender (13378ms)
- ✓ Should set new fee (219ms)
- ✓ Should set new fee precision (479ms)

When FULCRUM

- ✓ Should deposit [1] (3039ms)
- ✓ Should advanceBlockAtTime (123ms)
- ✓ Should deposit [2] (2463ms)
- ✓ Should increase pool size (2455ms)
- ✓ Should deposit [2] (4218ms)
- ✓ Should transfer [2 -> 3] (711ms)
- ✓ Should withdraw [1] (4297ms)
- ✓ Should withdraw [2] (3035ms)
- ✓ Should increase pool size (3243ms)
- ✓ Should withdraw [3] (4493ms)
- ✓ Fee should be payed (118ms)

Owner

- ✓ Should change lender (3415ms)
- ✓ Should call getPricePerFullShare (227ms)

When FORTUNE

- ✓ Should advanceBlockAtTime (106ms)
- ✓ Should deposit [2] (3136ms)
- ✓ Should increase pool size (3084ms)
- ✓ Should deposit [2] (13531ms)
- ✓ Should transfer [2 -> 3] (469ms)
- ✓ Should withdraw [1] (3774ms)
- ✓ Should withdraw [2] (3225ms)
- ✓ Should increase pool size (2162ms)
- ✓ Should increase pool size (4616ms)
- ✓ Fee should be payed (171ms)

Renounce Ownership



```
✓ should call renounceOwnership (359ms)
ERC20
✓ Should deploy (1123ms)
✓ Should call standart erc20 functions (1519ms)
```

Contract: BaseStrategy

Initialize

```
✓ Should deploy (16829ms)
```

Success

```
✓ Should call delegatedAssets (155ms)
✓ Should call harvestTrigger [when not activated] (1608ms)
✓ Should call harvestTrigger [when block.timestamp.
    sub(params.lastReport) >= maxReportDelay] (1931ms)
✓ Should call harvestTrigger [when outstanding
    > debtThreshold] (400ms)
✓ Should call harvestTrigger [when total.add(debtThreshold)
    < params.totalDebt] (456ms)
✓ Should call harvestTrigger [when total
    > params.totalDebt] (488ms)
✓ Should call harvestTrigger [when profitFactor.mul(callCost)
    < credit.add(profit)] (899ms)
✓ Should distributeRewards (2351ms)
```

Contract: xVault [2]

Initialize

```
✓ Should deploy (7470ms)
✓ Should setGovernance (400ms)
✓ Should setGovernance (453ms)
✓ Should setDepositLimit (459ms)
✓ Should setTreasury (620ms)
✓ Should setGuardian (388ms)
✓ Should setPerformanceFee (260ms)
✓ Should setManagementFee (331ms)
✓ Should addStrategy [0] (10901ms)
```

When !flashLoanActive

When deficit

```
✓ Should deposit { user: 0xCB0... } (1980ms)
✓ Should deposit { user: 0x129... } (1807ms)
✓ Should harvest { delay: 43200000 } (23942ms)
✓ Should withdraw { user: 0xCB0... } (854ms)
✓ Should setFlashLoan to false (770ms)
✓ Should deposit { user: 0xCB0... } (1206ms)
```

- ✓ Should deposit { user: 0x129... } (1090ms)
- ✓ Should harvest { delay: 43200000 } (11658ms)
- ✓ Should withdraw { user: 0xCB0... } (829ms)
- ✓ Should withdraw { user: 0x129... } (2189ms)

Contract: xVault [1]

Initialize

- ✓ Should deploy (4068ms)
- ✓ Should setGovernance (373ms)
- ✓ Should setGovernance (327ms)
- ✓ Should setDepositLimit (296ms)
- ✓ Should setTreasury (410ms)
- ✓ Should setGuardian (436ms)
- ✓ Should setPerformanceFee (270ms)
- ✓ Should setManagementFee (207ms)
- ✓ Should addStrategy [0] (2849ms)

Success

Strategy [0]

Setup strategy

- ✓ Should call apiVersion (137ms)
- ✓ Should call delegatedAssets (140ms)
- ✓ Should call name (172ms)
- ✓ Should setStrategist (284ms)
- ✓ Should setKeeper (227ms)
- ✓ Should setRewards (224ms)
- ✓ Should setMinXvsToSell (294ms)
- ✓ Should setMinWant (274ms)
- ✓ Should setCollateralTarget (365ms)

Investments

- ✓ Should call priceCheck (1086ms)
- ✓ Should deposit { user: 0xCB0... } (1331ms)
- ✓ Should deposit { user: 0x129... } (1383ms)
- ✓ Should trigger harvest/tend { delay: 43200000, tendTrigger: false, harvestTrigger: 7500000 } (5396ms)
- ✓ Should trigger harvest/tend { delay: 43200000, tendTrigger: false, harvestTrigger: 7500000 } (2070ms)
- ✓ Should withdraw { user: 0x129... } (854ms)
- ✓ Should deposit { user: 0x129... } (1468ms)
- ✓ Should withdraw { user: 0x129... } (963ms)
- ✓ Should call getblocksUntilLiquidation (460ms)

Change strategy

Owner

```

✓ Should addStrategy [1] (10472ms)
✓ Should removeStrategyFromQueue [0] (354ms)
Strategy [1]
✓ Should deposit { user: 0xCB0... } (1125ms)
✓ Should trigger harvest/tend { delay: 43200000,
  tendTrigger: false, harvestTrigger: false } (6126ms)
✓ Should withdraw { user: 0xCB0... } (809ms)
Owner
✓ Should addStrategyToQueue [0] (419ms)
✓ Should updateStrategyDebtRatio (451ms)
✓ Should updateStrategyRateLimit (256ms)
✓ Should updateStrategyPerformanceFee (259ms)
Strategy [0, 1]
✓ Should deposit { user: 0xCB0... } (1562ms)
✓ Should deposit { user: 0x129... } (929ms)
✓ Should call expectedReturn [0] (656ms)
✓ Should call creditAvailable [0] (186ms)
✓ Should call availableDepositLimit (179ms)
✓ Should call pricePerShare (158ms)
✓ Should call maxAvailableShares (224ms)
✓ Should call totalAssets (201ms)
✓ Should call getApy (125ms)
✓ Should call delegatedAssets [0] (131ms)
✓ Should setProfitFactor [0] (207ms)
✓ Should setDebtThreshold [0] (235ms)
✓ Should setMaxReportDelay [0] (244ms)
✓ Should trigger harvest/tend { delay: 43200000,
  tendTrigger: 7500000, harvestTrigger: 7500000 } (13749ms)
✓ Should withdraw { user: 0xCB0... } (1106ms)
✓ Should deposit { user: 0x129... } (1270ms)
✓ Should trigger harvest/tend { delay: 43200000,
  tendTrigger: 7500000, harvestTrigger: 7500000 } (9002ms)
Owner
✓ Should deploy strategy [2] (6024ms)
✓ Should tend [0] (574ms)
✓ Should setForceMigrate [0] (187ms)
✓ Should migrateStrategy [0 -> 2] (3444ms)
✓ Should setEmergencyExit [1] (3161ms)
✓ Should setWithdrawalQueue (428ms)
Strategy [2]
When !flashLoanActive
✓ Should setFlashLoan to false (770ms)

```

```

When !deficit
    ✓ Should deposit { user: 0xCB0... } (1117ms)
    ✓ Should deposit { user: 0x129... } (1212ms)
    ✓ Should trigger harvest/tend { delay: 43200000,
        tendTrigger: 7500000, harvestTrigger: 7500000 } (4044ms)
    ✓ Should withdraw { user: 0xCB0... } (926ms)
    ✓ Should withdraw { user: 0x129... } (853ms)

Owner
    ✓ Should deploy strategy [3] (8120ms)

Strategy [3]
When !flashLoanActive
    When !deficit
        ✓ Should deposit { user: 0xCB0... } (1359ms)
        ✓ Should deposit { user: 0x129... } (1358ms)
        ✓ Should harvest { delay: 43200000 } (3013ms)
        ✓ Should withdraw { user: 0xCB0... } (700ms)
        ✓ Should setFlashLoan to false (597ms)
        ✓ Should deposit { user: 0xCB0... } (1181ms)
        ✓ Should deposit { user: 0x129... } (1245ms)
        ✓ Should harvest { delay: 43200000 } (2260ms)
        ✓ Should withdraw { user: 0xCB0... } (799ms)
        ✓ Should withdraw { user: 0x129... } (906ms)

Owner
    ✓ Should deploy strategy [4] (5770ms)
    ✓ Should migrate strategy [3 -> 4] (4823ms)

When emergency shutdown
    ✓ Should deposit { user: 0xCB0... } (1380ms)
    ✓ Should deposit { user: 0x129... } (1128ms)
    ✓ Should setEmergencyShutdown (1711ms)
    ✓ Should harvest { delay: 43200000 } (2977ms)
    ✓ Should withdraw { user: 0xCB0... } (918ms)
    ✓ Should withdraw { user: 0x129... } (885ms)
    ✓ Should NOT deposit

When strategy receives wrong tokens
    ✓ Should setProtectedTokens (2125ms)
    ✓ Should send token to strategy (498ms)
    ✓ Should sweep token (878ms)

```

215 passing (8m)



We are delighted to have a chance to work together with Xend Finance team and contribute to their success by reviewing and certifying the security of the smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.