MixBytes()

PICKLE.FINANCE

SMART CONTRACT

AUDIT REPORT

(SCOPE 2)

NOVEMBER 12

2020

FOREWORD TO REPORT

A small bug can cost you millions. **MixBytes** is a team of experienced blockchain engineers that reviews your codebase and helps you avoid potential heavy losses. More than 10 years of expertise in information security and high-load services and 18 000+ lines of audited code speak for themselves. This document outlines our methodology, scope of work, and results. We would like to thank **Pickle.Finance** for their trust and opportunity to audit their smart contracts.

CONTENT DISCLAIMER

This report is public upon the consent of **Pickle.Finance**. **MixBytes** is not to be held responsible for any damage arising from or connected with the report. Smart contract security audit does not guarantee an inclusive analysis disclosing all possible errors and vulnerabilities but covers the majority of issues that represent threat to smart contract operation, have been overlooked or should be fixed.

TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

GENERAL PROVISIONS

Pickle.Finance is an experiment in using farming to bring stablecoins closer to their peg.

SCOPE OF THE AUDIT

The scope of the audit includes following smart contract at:

1. **strategy-cmpd-dai-v1.sol**
2. **crv-locker.sol**
3. **scrv-voter.sol**
4. **strategy-curve-3crv-v1.sol**
5. **strategy-curve-rencrv-v1.sol**
6. **strategy-curve-scrv-v3_1.sol**
7. **strategy-curve-scrv-v4.sol**
8. **strategy-curve-scrv-v4_1.sol**
9. **strategy-uni-eth-dai-lp-v3_1.sol**
10. **strategy-uni-eth-usdc-lp-v3_1.sol**
11. **strategy-uni-eth-usdt-lp-v3_1.sol**
12. **strategy-uni-eth-wbtc-lp-v1.sol**
13. **strategy-base.sol**
14. **strategy-curve-base.sol**
15. **strategy-staking-rewards-base.sol**
16. **strategy-uni-farm-base.sol**
17. **pickle-jar.sol**
18. **pickle-swap.sol**

The audited commit identifier is: **9b0f330a16bc35c964211feae3b335ab398c01b6**

02 SECURITY ASSESSMENT PRINCIPLES

CLASSIFICATION OF ISSUES

CRITICAL

Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

MAJOR

Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

WARNINGS

Bugs that can break the intended contract logic or expose it to DoS attacks.

COMMENTS

Other issues and recommendations reported to/acknowledged by the team.

SECURITY ASSESSMENT METHODOLOGY

Two auditors independently verified the code.

Stages of the audit were as follows:

1. “Blind” manual check of the code and its model
2. “Guided” manual code review
3. Checking the code compliance to customer requirements
4. Discussion of independent audit results
5. Report preparation

03 DETECTED ISSUES

CRITICAL

Not found.

MAJOR

Not found.

WARNINGS

1. Uncontrolled call of functions

crv-locker.sol#L84

Uncontrolled call of functions of another contract on behalf of this contract with the possibility of transferring ETH. Such a call can be made both by the user `governance` and by other users from the map `voters`.

In the contract **scrv-voter.sol#L150** and **scrv-voter.sol#L169**, there is a reference to such a possibility. But you can call other functions of this contract from any other contract. This can lead to unpredictable consequences.

strategy-base.sol#L196

The `timelock` user in the `_data` parameter can pass any value, which can lead to unpredictable consequences when accessing the contract with the `_target` address.

This particular issue qualified as a warning due to the security model assuming that voters are trusted persons/contracts.

Anyway we recommend limiting the use of such functionality because uncontrolled calls have very high risks.

Status:

ACKNOWLEDGED

Client's comment: The team is aware of this issue, but would like to state that this is necessary since Curve will not whitelist proxy contracts (i.e. we cannot easily change our implementation).

Use of the `execute` function is reserved for emergency situations. If it is used in an adversarial manner, there is a 12 hour timelock for participants to exit the system.

Beyond this, control of this function is handled by a 3 of 6 community controlled multisig wallet. The original team is no longer in majority control and power has been transferred to community participants to ensure good faith outcomes.*

2. No check of parameter values when executing smart contract constructor

It is recommended to append `require(variableAddress != address(0))` requirement right after the constructor execution starts.

strategy-curve-base.sol:

* line 45 variable `_curve`
* line 46 variable `_gauge`

strategy-staking-rewards-base.sol:

* line 22 variable `_rewards`

strategy-uni-farm-base.sol:

* line 36 variable `_token1`

crv-locker.sol:

* line 25 variable `_governance`

scrv-voter.sol:

* line 31 variable `_governance`
* line 32 variable `_crvLocker`

strategy-curve-scrv-v4.sol:

* line 88 variable `_governance`
* line 89 variable `_strategist`
* line 90 variable `_controller`
* line 92 variable `_scrvVoter`
* line 93 variable `_crvLocker`

strategy-curve-scrv-v4_1.sol:

* line 63 variable `_scrvVoter`
* line 64 variable `_crvLocker`

pickle-jar.sol:

* line 31, 32 variable `_token`
* line 33 variable `_governance`
* line 34 variable `_timelock`
* line 35 variable `_controller`

Status:

FIXED at **https://github.com/pickle-finance/protocol/pull/17/files***

3. Checking for the existence of the recipient's address

One cannot be sure that a third-party smart contract whose functions are accessed using the `IController(controller).treasury()` interface returns the correct address value.

In the following places, tokens are transferred using the safeTransfer function, but the existence of an address before the call is not checked:

* **strategy-base.sol#L151**
* **strategy-uni-farm-base.sol#L62**
* **strategy-uni-farm-base.sol#L97**
* **strategy-uni-farm-base.sol#L101**
* **strategy-uni-farm-base.sol#L111**

* **strategy-curve-3crv-v1.sol#L105**
* **strategy-curve-3crv-v1.sol#L129**

* **strategy-curve-rencrv-v1.sol#L87**
* **strategy-curve-rencrv-v1.sol#L111**

* **strategy-curve-scrv-v3_1.sol#L124**
* **strategy-curve-scrv-v3_1.sol#L155**

* **strategy-curve-scrv-v4.sol#L236**
* **strategy-curve-scrv-v4.sol#L324**
* **strategy-curve-scrv-v4_1.sol#L200**
* **strategy-cmpd-dai-v1.sol#L351**

In the following places, tokens are transferred using the safeTransfer function, but the existence of an address after the call is not checked

`IController(controller).devfund ()`:

* **strategy-base.sol#L147**
* **strategy-curve-scrv-v4.sol#L232**

Access to the functionality of smart contracts with tokens is carried out using interfaces. But not all token smart contracts can include logic to check addresses for zero values. Therefore, it is best to do this before accessing the interfaces. Better to do this in one place for all calls in the `src/lib/erc20.sol` library.

Status:

ACKNOWLEDGED

Client's comment: Changing this in the ERC20 library would force the function's implementation to deviate from the original OpenZeppelin source code. It also removes the ability to burn tokens by sending them to a zero address. Since we have implemented constructor checks for zero addresses, the risk of mistakenly sending tokens to a zero address is largely minimized.

4. No limit on the number of loop iterations

strategy-cmpd-dai-v1.sol#L287

With small `_borrowAndSupply` and `supplied` values and a large `_supplyAmount` value, there will come a point when there will be so many iterations that there is not enough gas to record changes. It is recommended to check for the expected number of iterations before starting the loop.

This problem will not stop the work of the smart contract, because in the `leverageUntil` function, the key parameter of the `_supplyAmount` cycle can be set as a parameter when calling this function, and by selecting the required value, it will be possible to execute the functionality.

strategy-cmpd-dai-v1.sol#L320

With small `_supplyAmount` and `_redeemAndRepay` values and a large `supplied` value, there will come a point when there will be so many iterations that there is not enough gas to record changes. It is recommended to check for the expected number of iterations before starting the loop.

This problem will not stop the work of the smart contract, because in the `deleverageUntil` function, the key parameter of the `_supplyAmount` cycle can be set as a parameter when calling this function, and by selecting the required value, it will be possible to execute the functionality.

Status:

ACKNOWLEDGED

Client's comment: The team is aware of this issue, but there is already a mitigation strategy. We have the option to change `colFactorLeverageBuffer` and/or `colFactorSyncBuffer` to be of a higher or lower value to avoid the out of gas problem. Additionally, we can leverage or deleverage until a specified supply amount with the `deleverageUntil` and `leverageUntil` functions. Based on the contract's current operation, it consumes 1.8 million gas for leveraging or deleveraging to the max, so there is already a substantial buffer from the max limit of 12 million gas.

5. Interface Consistency

This warning is about the `StrategyBase` interface consistency.

Since the `StrategyCurveSCRVv4_1` interface was changed with recent audit results, to keep the strategy interface consistent it is recommended to update all the other `StrategyBase`-interfaced strategies:

* **strategy-curve-3crv-v1.sol#L40**
* **strategy-curve-rencrv-v1.sol#L41**
* **strategy-curve-scrv-v3_1.sol#L46**
* **strategy-curve-scrv-v4.sol#L118**

Their `getMostPremium()` and `harvest()` functions signature are required to be updated to keep interface compliance and avoid wrecking business-logic using this interface with inconsistent asset index result type or with already known front-run issue.

Status:

FIXED at **https://github.com/pickle-finance/protocol/pull/19**

COMMENTS

1. Duplicate code

Duplicate code, i.e. using the same code structures in several places. Combining these structures will improve your code. The use of duplicate code structures impairs the perception of the program logic and can easily lead to errors in subsequent code edits. Duplicate code violates SOLID (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion) software development principles.

strategy-base.sol

```
line 92,  97, 102, 117, 122, 188
‘require(msg.sender == timelock, “!timelock”);’
Can make access modifier:
modifier onlyTimelock {
   require(msg.sender == timelock);
   _;
}

line 107, 112
‘require(msg.sender == governance, “!governance”);’
Can make access modifier:
modifier onlyGovernance {
   require(msg.sender == governance);
   _;
}

line 131, 139
‘require(msg.sender == controller, “!controller”);’
Can make access modifier:
modifier onlyController {
   require(msg.sender == controller);
   _;
}
```

strategy-cmpd-dai-v1.sol

```
line 208, 216, 226, 234
‘require(
   msg.sender == governance || msg.sender == strategist, “!governance”
);’
Can make access modifier:
modifier onlyGovernanceOrStrategist {
   require(
      msg.sender == governance || msg.sender == strategist, “!governance”
   );
   _;
}
```

crv-locker.sol

```
line 33, 38, 73,
require(msg.sender == governance, “!governance”);’
Can make access modifier:
modifier onlyGovernance {
   require(msg.sender == governance, “!governance”);
   _;
}

line 49, 56, 63, 68, 82
‘require(voters[msg.sender] || msg.sender == governance, “!authorized”);’
Can make access modifier:
modifier onlyVotersOrGovernance {
  require(voters[msg.sender] || msg.sender == governance, “!authorized”);
   _;
}
```

scrv-voter.sol

```
line 36, 41, 46
‘require(msg.sender == governance, “!governance”);’
Can make access modifier:
modifier onlyGovernance {
   require(msg.sender == governance, “!governance”);
   _;
}

line 55, 68, 77, 106, 137, 158
‘require(strategies[msg.sender], “!strategy”);’
Can make access modifier:
modifier onlyStrategies {
   require(strategies[msg.sender], “!strategy”);
   _;
}
```

strategy-curve-scrv-v4.sol

```
line 168, 173, 178, 183, 188, 193, 198
‘require(msg.sender == governance, “!governance”);’
Can make access modifier:
modifier onlyGovernance {
   require(msg.sender == governance, “!governance”);
   _;
}

line 214, 224, 248
‘require(msg.sender == controller, “!controller”);’
Can make access modifier:
modifier onlyController {
   require(msg.sender == controller, “!controller”);
   _;
}
```

pickle-jar.sol

```
line 46, 51
‘require(msg.sender == governance, “!governance”);’
Can make access modifier:
modifier onlyGovernance {
   require(msg.sender == governance, “!governance”);
   _;
}

line 56, 61
‘require(msg.sender == timelock, “!timelock”);’
Can make access modifier:
modifier onlyTimelock {
   require(msg.sender == timelock, “!timelock”);
   _;
}
```

Status:

ACKNOWLEDGED

Client’s comment: The team understands the comments, but disagrees with the suggested changes. Abstractions made solely for the sake of DRY-ness can often lead to readability and maintainability issues. The team feels that the current implementation is more explicit and makes for much more readable and less error-prone code with less indirection.

2. Variable not needed

crv-locker.sol#L87

The variable `success` will always be true. We recommend removing this variable.

Status:

FIXED at **https://github.com/pickle-finance/protocol/pull/18**

04 | CONCLUSION AND RESULTS

The smart contract has been reviewed and no critical and major issues have been found. Several suspicious places were spotted (marked as a warning). We also recommend refactoring the code.

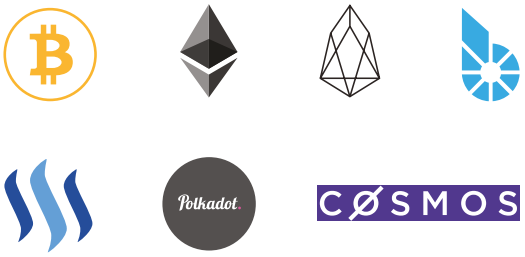
According to the implementation mostly consists of well-reviewed OpenZeppelin libraries (or modules implemented with following OpenZeppelin's manuals step by step) along with Uniswap interface modules, using extensive overflow prevention techniques with OpenZeppelin's `SafeMath` library, checking destination addresses along with message senders all the way, using reentrancy-safe implementation techniques, the contract itself is mostly safe to use.

ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, consult universities and enterprises, do research, publish articles and documentation.

Stack

Blockchains



JOIN US