SMART CONTRACT AUDIT (HTTPS://BLOG.COINFABRIK.COM/CATEGORY/SMART-CONTRACTS/SMART-CONTRACT-AUDIT-SMART-CONTRACTS/)

EasyPool Smart Contract Security Audit

Ariel Yabo (https://blog.coinfabrik.com/author/ariel-yabo/)

October 2, 2018 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/easypool-smart-contract-security-audit/)

easypool.io

Contents

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

Introduction

CoinFabrik has been hired to audit the EasyPool smart contracts.
We start this PDF report writing a summary with the smart contracts provided by the client and a list of the analysis methods used to audit the contracts. Next, we detailed our findings ordering the issues by severity, followed by all the observations we considered important to add. Furthermore, we ended up this audit report with a conclusion explaining how do the auditors value the code reviewed, and what are the most important things that need to be corrected to it to make it work flawlessly and securely.

Summary

The contracts audited are from the EasyPool repository at https://github.com/gitigs/easypool (https://github.com/gitigs/easypool). The audit is based on the commit ***c2e8b3c566da84071776ba02dc6f35f8e0dc0a4e***, and updated to reflect changes done on commit ***81c55b9fe2c437ec864ece8bc60cf9d5c6e4d3fa***

The audited contracts are:

- ProPool.sol: Pool management
- common/Affiliate.sol: Affiliate management for use in FeeService.sol
- common/CMService.sol: Service for managing the contracts
- common/FeeService.sol: Fee management service
- common/PoolFactory.sol: Pool deployment function
- common/PoolRegistry.sol: Pool registering
- common/Restricted.sol: Operator management functions
- library/ProPoolLib.sol: Pool manipulation functions

- library/QuotaLib.sol: Share claiming functions

The ProPool contract provides a public interface for managing the pool, the implementation of the functions being done on the ProPoolLib contract. This latter contract uses another library, defined in QuotaLib, to define an internal data structure within the contract's representation of an investment pool. There's also a dependency on a pair of functions from the FeeService contract.

As for CMService, it depends on the interface for PoolRegistry and PoolFactory, which inherit from the Restricted contract, adding operator management functions. PoolFactory also depends on ProPool, since it deploys new instances of the ProPool contract. CMService allows setting fee services, changing the pool factory contract, and such functions. PoolRegistry brings a function to register new pools, emitting an event.

Fortunately, on analysis of these contracts only a few minor issues, mostly with maintainability, were found. We proceed to detail the checks we've made, the improvements to the contracts which could help development, and a conclusion.

As for the audit, the following analyses were performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front-running.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
- Contract soft locking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Missing function qualifiers and their misuse.
- Fallback functions with a higher gas cost than the one that a transfer or send call allow.
- Fraudulent or erroneous code.
- Code and contract interaction complexity.
- Wrong or missing error handling.

    - Overuse of transfers in a single transaction instead of using withdrawal patterns.
(https://blog.coinfabrik.com/)
- Insufficient analysis of the function input requirements.

(https://blog.coinfabrik.com/)

File Summary

ProPool.sol

No vulnerabilities were found in this contract, but some compiler warnings were noted. This is a contract that provides public-facing presale pool management functions, which are implemented in the ProPoolLib library.

Affiliate.sol

A lack of use of SafeMath was noted, but no issues were found. This contract has functions for controlling affiliates within the contract.

CMService.sol

No vulnerabilities were found, as this contract was simple and straighforward. It’s a contract to which it can be called to set the addresses of the fee service, the pool factory, the pool registry and to deploy the ProPool contract by a call to poolFactory. This way the implementation can be changed without redeploying.

FeeService.sol

This contract does not use the SafeMath library consistently, but no major security concerns were found. The use of this contract is providing fee settings, which are then attached to the pool through the functions provided in CMService.sol

PoolFactory.sol

No vulnerabilities were found in this contract. Since it provides a simple function for the deploying of a new ProPool contract, no vulnerabilities are considered as possible.

PoolRegistry.sol

No vulnerabilities were found in this contract. It’s a simple contract providing a function that emits an event when called, recording the pool address and details.

Restricted.sol

No vulnerabilities were found here. This is an implementation providing storage and functions that manage operators, which are used to restrict function execution in PoolFactory.sol, PoolRegistry.sol and Affiliate.sol

ProPoolLib.sol

The inconsistent use of SafeMath was noted, but no major issues were found. This contract is the one providing the implementation that ProPool.sol uses.

QuotaLib.sol

Inconsistent SafeMath usage was noted, along with a lack of documentation. Other than that, the contract was straightforward, without much complexity. This contract is used to manipulate the refund and token quotas in the ProPoolLib.sol contract.

Detailed findings

Critical severity

None found.

Medium severity

None found.

Minor severity

Warnings on compilation

The contracts emit warnings when compiled. It would add to the auditability of the contracts themselves if they were removed. While they may not cause any problem directly, they can occlude other more important warnings. We recommend fixing these before deployment, mostly as a way of ensuring that everything's going well.

These warnings mostly concern uninitialized storage pointers, which are by default zeroed. However, these aren't considered as severe as they are later initialized correctly.

Inconsistent usage of SafeMath

There's not enough usage of SafeMath in some lines in QuotaLib.sol, yet it is declared as being used in the following statement:

```
library QuotaLib {

using SafeMath for uint;
```

We recommend either removing the statement or following through in the usage consistently since this could lead to errors and possible future changes could cause bugs. We also recommend the use of SafeMath in the other contracts, which it can be seen that it was considered yet decided against in some commented out parts of the code, which declare

```
// We could use SafeMath to catch errors in the logic,

// but if there are no such errors we can skip it.

// using SafeMath for uint;
```

This is something which was found in ProPoolLib.sol, in Affiliate.sol and in FeeService.sol. While it is true that SafeMath has no purpose if the code is correct, it can act as a safeguard if that isn't the case. It'd be advisable for the team to reconsider this design decision since it'd mean arithmetical operations are guarded against overflow.

Enhancements

Not enough documentation

In the QuotaLib.sol contract there's not enough documentation. It'd be advisable for the development team to fix this, as it helps auditability and legibility of the contracts. There are many comments of the form:

```
/**
* @dev ...
*/
```

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

Changing these to have meaningful commentary can avoid errors when handling the contract, so it's suggested to follow through.

Old compiler pragmas

The contracts have the following pragma directive:

```
pragma solidity ^0.4.24;
```

We remind the development team that the Solidity compiler has been updated. While this isn't critical in these particular contracts, there are bugfixes which if absent could've caused problems. As usual, it is recommended to stay on top of the new versions, which could avoid problems with things like the ABI encoding, parser issues, et cetera.

Observations

We annotate some of the verifications done to the contracts to indicate that they were performed even when no critical issue was found during the audit.

- Misuse of the different call methods: *call.value()*, *send()* and *transfer()*.
  We found no incorrect use of *call()* or *send()*, and *transfer()*. Calls to these methods were done on checked input, with correct coverage of requirements.
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Race conditions such as reentrancy attacks or front-running.
    - The only call to an external contract is in the *addressCall* function, but considering it's private and is only called from *payToPresale*, which calls to this function with the *presaleAddress* that's set. Since this is only callable by the pool administrator and the state change of the pool is done before the call, the pool will have a consistent internal state, so reentrancy can't cause damage.
  - There are many setters for addresses that could if not enough checks are done on input, cause transfers to unwanted addresses. However, these functions

have the correct modifiers so only the administrator of the pool can change these.

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

- No use of block timestamps, assuming things other than them being strictly increasing.
  - Timestamps aren't used in the contract at all, so attacking the contract from that angle is impossible.

- Contract softlocking attacks (DoS) / unbounded gas usage.
  No function in the contract has a loop that can be abused to cause a soft lock or an unbounded usage of gas.

Conclusion

We found the contracts to be simple and straightforward. No critical issues were found, yet some possible legibility issues were considered. Overall the contracts had decent documentation, except for some points, but in general, they were simple enough to follow that it wasn't critical. There is some code which we recommend reviewing, but in general, the state is good, the only trouble being somewhat inconsistent snippets which reduce legibility, which could lead to bugs if additions are made.

***Disclaimer**: This audit report is not a security warranty, investment advice, or an approval of the EasyPool project since Coinfabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.*

Related Posts

(https://blog.coinfabrik.com/smart-contracts/etherparty-token-smart-

EtherParty Smart Contract Security Audit (https://blog.coinfabrik.com/smart-contracts/etherparty-token-smart-contract-security-audit-coinfabrik/)

contract-
security-
audit-
coinfabrik/)

Coinfabrik team has been hired to audit Etherparty smart contracts. Firstly, we will provide a…

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/smart-
contracts/smart-
contract-
audit-
smart-
contracts/bricks4us-
token-
smart-
contract-
security-
audit/)

Bricks4US Token Smart Contract Security Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/bricks4us-token-smart-contract-security-audit/)

CoinFabrik was asked to audit the contracts for the Bricks4Us token sale. Firstly, we will…

(https://blog.coinfabrik.com/smart-
contracts/smart-
contract-
audit-
smart-
contracts/rcn-
basiccosigner-

RCN BasicCosigner Smart Contract Security Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/rcn-basiccosigner-smart-contract-security-audit/)

Coinfabrik security audit's team was asked to audit the BasicCosigner contract.

smart-
contract-
security-
audit/)

This contract is part...

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/smart-
contracts/smart-
contract-
audit-
smart-
contracts/cryptosolartech-
security-
audit/)

Cryptosolartech Security Audit
(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/cryptosolartech-security-audit/)

Coinfabrik's smart contract audit's team was asked to audit the contracts for the Cryptosolartech sale....

SHARE ON

(https://www.facebook.com/sharer/sharer.php?u=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/easypool-smart-contract-security-audit/)
(https://twitter.com/intent/tweet?text=EasyPool%20Smart%20Contract%20Security%20Audit&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/easypool-smart-contract-security-audit/)
(https://pinterest.com/pin/create/button/?url=&media=https://blog.coinfabrik.com/wp-content/uploads/2019/04/easypool.png&description=EasyPool+Smart+Contract+Security+Audit)
(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/easypool-smart-contract-security-audit/&title=EasyPool%20Smart%20Contract%20Security%20Audit&source=CoinFabrik%20Blog)

← PREVIOUS ARTICLE

A Summary of Satis Group's Latest Cryptoasset Valuation Report (https://blog.coinfabrik.com/finance/a-summary-of-satis-groups-latest-cryptoasset-valuation-report/)

NEXT ARTICLE →

Blockchain Mediated Prediction Markets (https://blog.coinfabrik.com/finance/blockchain-mediated-prediction-markets/)

You may also like

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/)

(https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

Magic Bridge Audit (https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

2 months ago

Smart Contract Audit (https://blog.coinfabrik.com/category/smart-contracts/smart-contract-audit-smart-contracts/)

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

(https://blog.coinfabrik.com/feed/)

(https://ar.linkedin.com/company/coinfabrik)

(https://blog.coinfabrik.com/)

(https://twitter.com/coinfabrik)

(https://blog.coinfabrik.com/)

(https://www.youtube.com/channel/UC2GmjCr7aEz-il31kqOy9aw)

(https://www.facebook.com/CoinFabrik/)

(https://www.reddit.com/r/CoinFabrik/)

(https://github.com/coinfabrik)