BITBANK

This report contains the results of our engagements with bitbank, Inc. in order to review a series of smart contracts that allow for the management of deposits of Ethereum-based assets at the bitbank exchange.

1 | EXECUTIVE SUMMARY

This report contains the results of our engagements with bitbank, Inc. in order to review a series of smart contracts that allow for the management of deposits of Ethereum-based assets at the bitbank exchange.

John Mardlin, Nicholas Ward, and Nicholas Ward conducted the review over a week. It lasted from November 9th through 13th, 2020. Total of 10 person-days were used.

We reviewed the code manually and used fuzzing tests (see Appendix 1) during the engagement.

We were impressed with the way the system was designed and tested. The clients' goals were also clearly defined.

2 | SCOPE

Our review initially focused on the commit hash d0e884fae6a8342f5464086db0d8205479b0ef78. We also reviewed several pull requests, which resulted in a final commit hash of 3daeedb029ef4642e82550a24726875675284bb3.

2.1 | Objectives

Following is a list of objectives from the client for the contract system. The review included confirmation that the system meets these objectives.

1. Optimize gas consumption for the depositor proxy.
2. Optimize gas consumption for the ETH deposit-to-deposit address path.
3. 3. Allow the ExchangeDeposit contract to be killed. This should cause future ETH deposits not to succeed.
4. Make it possible to support ERC20 deposits and allow them to be moved at the coldAddress. This should cost in the 50k range for gas.
5. If the contract is terminated, the ERC20 deposits retrieved should be sent to adminAddress.

3 | RECOMMENDATIONS

3.1 | Move `deployNewInstance()` into a separate factory contract

Description

The `ExchangeDeposit` `deployNewInstance()` function can be used to deploy proxy depositor contracts. These proxies should be deployed using a separate factory contract in order to simplify the core logic and reduce attack surface.

3.2 | Replace `exchangeDepositorAddress()`

Description

The function `exchangeDepositorAddress()` is used for two purposes:

1. 1. Find the `ExchangeDeposit` address where mutable values, such as `coldAddress`, are stored.

2. To determine the context for the current call.

This introduces unnecessary complexity, and could be replaced by an address `immutable exchangeDeposit` amount set in the constructor. This would make the `ExchangeDeposit` address part of the deployed binary code.

This is used to enforce important properties, such as the restriction on storage writes within the `ExchangeDeposit` contract context. It can lead to incorrect assumptions about the caller or call context, and it could be replaced with a simpler check.

`ExchangeDeposit` should not compare the bytecode at `address(this)`, to that of the depositor proxy. Instead, `ExchangeDeposit` should match `address(this)`, to the immutable address mentioned above. This allows the contract determine if its code is running within its own address context.

Recommendation

By addressing the two uses of `exchangeDepositorAddress()` with simpler methods, the amount of low-level assembly can be reduced significantly, invalid assumptions about the source and context of the call can be avoided, and the depositor proxy implementation can be decoupled from the `ExchangeDeposit` implementation.

3.3 | Gas will be saved more if there are more optimizer rounds

A small amount of gas can be saved by increasing the number of optimizer run in truffle-config.js for common operations.

3.4 Separate `ExchangeDepositor ()` into 2 functions

The name is `ExchangeDepositor()` suggests that it should return a boolean, but it also returns the `exchangeDepositorAddress`. This should make it easier to understand the logic.

This function can be simplified by comparing `address(this)`, to an `immutable type of address` that was determined during deployment.

4 | SECURITY SPECIFICATION

This section describes the security perspective of the system being audited. This section is not intended to replace documentation. This section identifies security properties that have been validated by the audit team.

4.1 Actors

Below are the relevant actors and their abilities:

Admin: Admin: One admin address is created in the constructor. If the ExchangeDeposit is "alive", the Admin can take the following actions (coldAddress!= 0x0):

| Change the coldAddress
| Modify the address of implementation
| Modify the minimumInput
| Call kill() to stop standard deposits and other admin actions in the future

Depositor: A proxy contract is created for each depositor. It contains the address of the main ExchangeDeposit implementation that will be the recipient of any calls. The following actions can be taken by a depositor proxy:

| ExchangeDeposit can be reached at (855) 882-7880 to make a deposit of ETH
| DELEGATECALL is to any function that has not been modified by

4.2 Trust Model

It is important to establish trust between actors in any system. The following trust model was created for this audit:

| The Admin is trusted by depositors with the ability to block or redirect deposits, adjust minimum deposit amounts, and run arbitrary codes within the context of any proxy depositor contract.

This trust level is high, but it is consistent with the trust that comes from using a central exchange.

4.2 Trust Model

This is not an exhaustive list of security properties that were checked during this audit.

| The depositor:

| Calls or delegatecalls no other address than the target address in its bytecode.
| ExchangeDeposit will only accept calls if calldata is not provided.
| ExchangeDeposit can delegatecall if and only when at least one byte of calldata has been provided.

| ExchangeDepositor does not store any storage writes except in functions that can be called only by adminAddress.
| It is impossible to reverse the "killed" state (coldAddress ==0x0).
| ExchangeDepositor's existing functions cannot be modi(ed to alter the storage of a proxy depositor whose target is main.

| ExchangeDepositor example.

| This property is only valid for the ExchangeDepositor we reviewed. However, the admin could set an implementation address that can write to the proxy storage.

5 | FINDINGS

Each issue is assigned a severity:

- **Minor** problems are subjective. These are usually suggestions about best practices or readability. These issues should be addressed by code maintainers.
- **Medium** issues are objective, but they are not security vulnerabilities. These issues should be addressed, unless there are compelling reasons not to.
- Security vulnerabilities are critical issues that can't be exploited directly or require special conditions to be exploited. All of these **Major** problems should be addressed.
- Security vulnerabilities that could be exploited to cause **Critical** issues need to be addressed.

5.1 | ERC20 tokens with no return value will fail to transfer Major Fixed

This issue was solved using OpenZeppelin's SafeERC20.

Description

Many tokens do not comply with the ERC20 standard, even though it suggests that a successful transfer should return `true`.

If that happens, the `.transfer()` function here will return even if the transfer succeeds, since solidity will verify that the RETURNDATASIZE matches that of the ERC20 interface.

code/contracts/ExchangeDeposit.sol:L229-L231

```
if (!instance.transfer(getSendAddress(), forwarderBalance)) {
    revert('Could not gather ERC20');
}
```

Recommendation

Consider using OpenZeppelin's SafeERC20.

APPENDIX 1 - FUZZ TESTING OF DEPOSITOR PROXY

In addition to manual inspection of the code, the correctness of the bytecode implementation of the depositor proxy contract was assessed using Harvey, our in-house greybox fuzzer for smart contracts (see https://arxiv.org/pdf/1905.06944.pdf for more details). The proxy was fuzzed using a custom testing harness for 15 hours. This resulted in more than 2 million unique testing inputs.

Important security properties of each proxy were tested for each input. During the fuzzing campaign, no violations of these properties was detected.

This method of testing allowed the behavior of proxy contracts to be evaluated using a large number inputs. However, it is important that you understand its limitations. When evaluating the results of this analysis, it is important to consider the following:

| It is obvious that not all paths could be explored, especially when it comes to external calls made by proxy.
| Given time constraints, there are only a few properties that can be checked.
| There is always the possibility of an error in the fuzzing harness or assertions of important property.

APPENDIX 2 DISCLOSURE

ConsenSys Dialigence ("CD") receives compensation from clients (the Clients) for performing the analysis in these reports (the Reports). Reports can be distributed via ConsenSys publications or other distributions.

Reports do NOT endorse or indict any project or team. They also don't guarantee security for any project. This Report doesn't consider or have any bearing on the economics of token sales, token tokens, or any other product, services, or assets. Cryptographic tokens, which are emerging technologies, carry high technical risks and uncertainties. Any Report does not provide any representation or warranty to Third-Parties in any way. This includes regarding the bug-free nature of code, any business model or proprietors, or the legal compliance of such businesses. The Reports should not be relied upon by any third party, even if it is used to make decisions about buying or selling tokens, products, services, or assets. This Report is not intended as investment advice and should not be relied on as such. It is also not an endorsement of this team or project, and is not a guarantee of absolute security. CD is not obligated to any Third-Party for publishing these Reports.

PURPOSE OF THE REPORTS Reports and analysis contained therein are only for Clients. They can be published only with their permission. Our review will only cover Solidity code. We are limited to reviewing the Solidity codes we have identified as being included in this report. Solidity language is still under development. It may have flaws and risks. The review does NOT cover the compiler layer or any other areas that could pose security risks beyond Solidity. Cryptographic tokens, which are emerging technologies, carry high technical risk and uncertainty.

CD makes the Reports accessible to clients and other parties (i.e. "third parties") via its website. CD hopes that the public availability of these analyses will help the blockchain ecosystem to develop best practices in this rapidly changing area of innovation.