Smart Contract Audit

Date	September 2019

ATOMIC LOANS

ConsenSys Diligence performed a second security audit of the Atomic Loans smart-contract system.

1 | SUMMARY

ConsenSys Diligence performed a second security audit of the Atomic Loans smart-contract system. The original report can be found here: https://github.com/ConsenSys/atomic-loans-audit-report-2019-07. For a detailed explanation and overview of the system, please refer to the original report.

2 | CHANGES COVERED IN THIS AUDIT

This audit focuses on the following major changes:

1. Compound was introduced to the Funds contract. Lenders now have the option to have their funds earn interest via Compound even if they haven't been lent.
2. All loans that default are subject to a global default parameter. A globally adjustable global interest rate is included. These parameters can be used by "custom" loans.
3. Liquidation sales no longer count as auctions. Instead, collateral is purchased at a 7% discount according to the oracles.
4. To circumvent a Bitcoin script limit, the collateral swap is done using a slightly different protocol. The new protocol requires that the collateral be moved to a P2SH first before any "back" signatures can be created. To complete the collateral sale, this means that the buyer must provide another secret.

3 | AUDIT SCOPE

The system was evaluated by the audit team as being secure, resilient, and operating according to its specifications. These are the three main categories that can be broken down into audit activities:

1. **Security:** Identification of security-related issues in the contract.
2. **Architecture:** Examining the system architecture using established smart contract best practices.
3. **Code quality:** An in-depth review of the contract source codes. These are the main areas of concern:
   - | Correctness
   - | Readability
   - | Scalability
   - | Complexity of code
   - | Test coverage quality

4 | 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 directly exploited or may need to be accessed under certain conditions. All of these **Major** problems should be addressed.
- Security vulnerabilities that could be exploited to cause **Critical** issues need to be addressed.

4.1 | Intentional secret reuse can block borrower and lender from accepting liquidation payment Major Fixed

This is fixed in AtomicLoans/atomicloans-eth-contracts#65.

Description

Dave (the liquidator), must reveal secret D in order to claim the collateral he has purchased. After that, Alice (the borrower) and Bob (the lender) can claim their payment.

Secrets must be provided via the `Sales.provideSecret()` function:

code/ethereum/contracts/Sales.sol:L193-L200

```
function provideSecret(bytes32 sale, bytes32 secret_) external {
        require(sales[sale].set);
        if      (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashA) { secretHashes[sale].secretA = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashB) { secretHashes[sale].secretB = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashC) { secretHashes[sale].secretC = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashD) { secretHashes[sale].secretD = secret_; }
else                                                                          { revert(); }
}
```

Note that if Dave chooses the same secret hash as either Alice, Bob, or Charlie (arbiter), there is no way to set secretHashes[sale].secretD because one of the earlier conditionals will execute.

To receive payment later, Alice and Bob must be able provide Dave's secret.

code/ethereum/contracts/Sales.sol:L218-L222

```
function provideSecret(bytes32 sale, bytes32 secret_) external {
        require(sales[sale].set);
        if      (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashA) { secretHashes[sale].secretA = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashB) { secretHashes[sale].secretB = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashC) { secretHashes[sale].secretC = secret_; }
else if (sha256(abi.encodePacked(secret_)) == secretHashes[sale].secretHashD) { secretHashes[sale].secretD = secret_; }
else                                                                          { revert(); }
}
```

Note that if Dave chooses the same secret hash as either Alice, Bob, or Charlie (arbiter), there is no way to set secretHashes[sale].secretD because one of the earlier conditionals will execute.

To receive payment later, Alice and Bob must be able provide Dave's secret.

code/ethereum/contracts/Sales.sol:L218-L222

```
function accept(bytes32 sale) external {
require(!accepted(sale));
require(!off(sale));
        require(hasSecrets(sale));
        require(sha256(abi.encodePacked(secretHashes[sale].secretD)) == secretHashes[sale].secretHashD);
```

Dave can use this to get the collateral free of charge:

1. Dave examines Alice's secret hashes in order to determine which will be used for the sale.
2. Dave uses the same secret hash to begin the liquidation process.
3. Alice and Bob share their secrets A andB during the move of the collateral.
4.Dave now has the preimage of the secret hash that he gave. Alice had already revealed it.
5. Dave uses this secret to get the collateral.
6. Alice and Bob want to be paid, but are unable to give Dave's secret to their `Sales` smart contract because of the order in `provideSecret()`.
7. Dave can request a refund after an expiration.

Mitigating Factors

Alice and Bob might notice that Dave selected a duplicate secret hash, and decline to sell the item. They are unlikely to do this.

Recommendation

You can either change the way `provideSecret()` operates to allow duplicate secret hashes, or reject duplicate hashes when you use `create()`.

4.2 | There is no way to convert between custom and non-custom funds

Medium Wont (x

If users want to change between non-custom and custom funds, they can create a new account. It is not a major burden since lenders will need to use agent software to manage funds. This work)ow involves creating a new address, as the private key must be given to agent software.

Description

Each fund is created using either Funds.create() or Funds.createCustom(). Each fund can only be created using Funds.create() or Funds.createCustom().

code/ethereum/contracts/Funds.sol:L348-L355

```
function create(
    uint256 maxLoanDur_,
    uint256 maxFundDur_,
    address arbiter_,
    bool    compoundEnabled_,
    uint256 amount_
) external returns (bytes32 fund) {
    require(fundOwner[msg.sender].lender != msg.sender || msg.sender == deployer); // Only allow one loan fund per address
```

code/ethereum/contracts/Funds.sol:L383-L397

```
function createCustom(
    uint256 minLoanAmt_,
    uint256 maxLoanAmt_,
    uint256 minLoanDur_,
    uint256 maxLoanDur_,
    uint256 maxFundDur_,
    uint256 liquidationRatio_,
    uint256 interest_,
    uint256 penalty_,
    uint256 fee_,
    address arbiter_,
    bool    compoundEnabled_,
    uint256 amount_
) external returns (bytes32 fund) {
    require(fundOwner[msg.sender].lender != msg.sender || msg.sender == deployer); // Only allow one loan fund per address
```

These functions are where bools[fund].custom can be set. Once a fund is created, there's no way of deleting it. It is impossible for an account to switch between custom and non-custom funds.

This can be problematic if the default parameters change in such a way that users find unappealing. Users may wish to change to using a custom funds but are unable to do this without creating a new Ethereum account.

Recommendation

You can either allow funds to disappear or allow funds that are custom to be changed between non-custom and custom.

4.3 | Funds.maxFundDur has no effect if maxLoanDur is set

Medium Fixed

This is (xed in AtomicLoans/atomicloans-eth-contracts#68.

Description

`MaxFundDur` is the maximum time that a fund can be active. This is checked in `request()` to make sure the loan's duration doesn't exceed the time. However, the check is skipped if `maxLoanDur` has been set.

code/ethereum/contracts/Funds.sol:L510-L514

```
if (maxLoanDur(fund) > 0) {
    require(loanDur_       <= maxLoanDur(fund));
} else {
    require(now + loanDur_ <= maxFundDur(fund));
}
```

Examples

A user can set `maxLoanDur` (the maximum amount of the loan) to one week and `maxFundDur` to December 1, which will result in a loan that expires on December 7.

Recommendations

Even if `maxLoanDur` has been set, check against `maxFundDur`.

4.4 | In Funds, maxFundDur is misnamed

Minor Fixed

This is (xed in AtomicLoans/atomicloans-eth-contracts#66.

Description

Funds.update() allows you to update the following fields. These fields are only available if bools[fund].custom has been set:

| minLoanamt
| maxLoanAmt
| minLoanDur
| interest
| Penalty
| Fee
| liquidationRatio

These changes will not take effect if bools[fund].custom has not been set. This could be misleading for users.

Examples

code/ethereum/contracts/Funds.sol:L454-L478

```
function update(
    bytes32  fund,
    uint256  minLoanAmt_,
    uint256  maxLoanAmt_,
    uint256  minLoanDur_,
    uint256  maxLoanDur_,
    uint256  maxFundDur_,
    uint256  interest_,
    uint256  penalty_,
    uint256  fee_,
    uint256  liquidationRatio_,
    address  arbiter_
) external {
    require(msg.sender == lender(fund));
    funds[fund].minLoanAmt       = minLoanAmt_;
    funds[fund].maxLoanAmt       = maxLoanAmt_;
    funds[fund].minLoanDur       = minLoanDur_;
    funds[fund].maxLoanDur       = maxLoanDur_;
    funds[fund].maxFundDur       = maxFundDur_;
    funds[fund].interest         = interest_;
    funds[fund].penalty          = penalty_;
    funds[fund].fee              = fee_;
    funds[fund].liquidationRatio = liquidationRatio_;
    funds[fund].arbiter          = arbiter_;
}
```

Recommendation

Two update functions could be created to address this issue: one for custom and one for other funds. These values can only be set by the update for custom fund.

APPENDIX 1 - 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 sales, 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 intended to be used as such as an endorsement of the project or its team. Furthermore, it does not guarantee 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 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.

LINKS TO OTHER WEBSITES FROM THIS WEB site You can, via hypertext or other computer hyperlinks, gain access web sites owned by people other than ConsenSys. These hyperlinks are provided only for your convenience and are not intended to replace the owners of these web sites. ConsenSysys or CD are not responsible or liable for any content or operation of these Web sites. You also agree that ConsenSysys or CD will not be liable for any third-party Web site. Except as stated below, linking from this Web Site to another site does not mean or imply that ConsenSysys or CD endorses that site's content or its operator. It is up to you to decide whether or not you can use content from any other websites to which the Reports link. ConsenSys or CD will not be responsible for third-party software used on the Web Site. They also assume no liability for any errors or inaccuracies of any output generated by such software.

TIMELINESS CONTENT. The Reports are current as of the Report's date. However, they can be modified at any time. ConsenSys or CD are the only sources of information, unless otherwise indicated.