Dharma Audit

OPENZEPPELIN SECURITY | JULY 31, 2018 Security Audits

The Dharma team asked us to review and audit the smart contracts implementing their protocol. We looked at the code and now publish our results.

The audited code is located in the dharmaprotocol/charta repository. The version used for this report is commit `b110959477cf37375bf7e9344d40eb85219c8575`.

Here is our assessment and recommendations, in order of importance.

Update: *The Dharma team has followed most of our recommendations and updated their contracts. The new version is at commit* `047327e8285b0a5f157c6c6a03ec142899c7572d`*. There are additional features at this version which have not yet been audited by us.*

Critical Severity

Anybody can mint DebtToken

`DebtToken` extends `MintableNonFungibleToken` and thus inherits a public `mint` function that allows the caller to create new tokens.

Given that there is an additional `create` function defined in `DebtToken` which requires authorization, it's clear to us that `mint` should require the same. Otherwise, it can be called by anyone with an arbitrary token id.

to be made. The entire system can be put at risk in this way.

To fix this, add to `mint` the same authorization check seen in `create`.

Update: *Fixed in `97690a9` by making `mint` an internal function in `MintableNonFungibleToken`.*

High Severity

Use of block numbers to express moments in time

Implementors of the `TermsContracts` interface are expected to define repayment terms as a function of block numbers, as documented and seen in the function signatures of `getExpectedRepaymentValue` and `getValueRepaid`. To aid in this, entries in `DebtRegistry` record the issuance block number.

Block timestamps have a small risk of miner manipulation within a range of minutes, so it is recommended to use block numbers for certain usecases where this manipulation is a security risk. It should be noted, however, that block times are not fixed, and thus block numbers are not a reliable unit of time. For example, the constant `SEVEN_DAYS_IN_BLOCKS` will not always represent the timespan of seven days that it is meant to.

For the specific use case of determining repayment dates we think that the predictability of timestamps is preferable, and that it does not compromise the security of the system. Consider changing interfaces and data structures to use timestamps instead of block numbers.

Update: *Changed to timestamps in `d4dc030`.*

Medium Severity

Unbounded loop in PermissionsLib

The `revokeAuthorization` function in `PermissionsLib` performs a linear search over an array of unbounded size. Through subsequent calls to `authorize`, the array could grow to a size so large that a call to `revokeAuthorization` would be prohibitively expensive and not fit in a block.

Update: *Fixed across several commits.*

Duplication of owner logic in DebtToken

`DebtToken` extends `NonFungibleToken` and redefines the latter's `setTokenOwne` and `ownerO` internal functions, so that they modify the `DebtRegistry` instead of the NFT's own tracking of ownership. This seems to result in coherent behavior, but we think it's an unnecessary meddling with `NonFungibleToken`'s semantics, and could eventually cause some obscure hard-to-spot issues.

We would suggest to remove the redefinition of `_ownerOf`, and to modify `_setTokenOwner` so that it runs `super._setTokenOwner()` additionally to the calls on the registry. In this way the registry is kept in sync with the token's tracking of ownership, but merely by intercepting changes of ownership, instead of entirely replacing that part of the token's implementation.

Update: *Fixed in* `7a618cf`.

DebtRegistry assumption could be broken

The contracts in the system assume that a valid `Entry` in the `DebtRegistry` will never have a null address for `beneficiary`. This assumption is used, for example, in `RepaymentRouter`'s `repay` to ensure a repayment is done for a valid issuance, as well as in `DebtRegistry` itself to check that an inserted entry is new and doesn't clash with a previous one.

An authorized agent could, however, insert an entry with a null beneficiary through `insert`, or set the beneficiary of an existing entry to the null one through `modifyBeneficiary`. Thus, the assumption can be broken.

Add checks in `insert` and in `modifyBeneficiary` to reject null beneficiaries.

Update: *Fixed in* `d314446`.

External query gas limit may be problematic

`EXTERNAL_QUERY_GAS_LIMIT`. (Although `getAllowance` doesn't limit the gas, a comment implies that it should be doing so.)

The `EXTERNAL_QUERY_GAS_LIMIT` constant has a value of `4999`, justified by the fact that changes to state require at least 5000 gas. Although we share the concern that an external call such as a balance check could cause a re-entrancy attack, the reality is that gas limits are not part of the ERC20 spec, and a token is free to implement a balance check that costs more than 4999 gas. In fact, quick tests show us that a balance check in a recently cloned MiniMe token (a popular token implementation) costs around 4320 gas, which is dangerously close to the constant limit imposed.

We would recommend to look into the recent new opcode `STATICCALL` to use as an alternative to limiting gas. This opcode will eliminate the risk of re-entrancy by ensuring that an external call performs no state changes, which is fine to assume for ERC20 functions `balanceOf` and `allowance`, as they are constant functions.

Update: *Due to the unavailability of STATICCALL in high-level Solidity code as of yet, the team has decided to stick with a simple gas limit. However, the limit was raised to 8000 and the missing check was added in* `ff1a22e`.

SimpleInterestTermsContract will silently accept other tokens

The function `registerRepayment` implemented in `SimpleInterestTermsContract` is meant to be called by the `RepaymentRouter` whenever a debtor repays part of an issued debt. Although said terms contract requires repayments to be done in a specific token `repaymentToken`, `registerRepayment` could be called with a different token as parameter. This is recognized by the contract, since the `valueRepaid` is only updated when the correct token is used, but the function will silently accept any other token. We would recommend to reject any other token, by returning `false` from `registerRepayment`.

Update: *Fixed in* `1bfa459`.

Mismatch between signature and implementation of getValueRepaid

and `NFTTermsContract` do not implement this semantics: they ignore the `blockNumber` argument and return the value corresponding to the moment when the function is called. Consider either changing the interface and documentation, or implementing the documented behavior.

Update: *Fixed in* `6db0eee`.

Low Severity

Inconsistent Pausable operations in DebtRegistry and DebtToken

The main functionality of `DebtRegistry` is correctly guarded with `whenNotPaused` modifiers to ensure they cannot be used when the contract is in "paused" state. There are other non-constant functions in the contract (those related to permissions) which are inconsistently guarded: `revokeInsertAgentAuthorization` has the `whenNotPaused` modifier, but none of the others do. Consider either removing the modifier in this function or adding it to the other three permissions-related functions in the contract for consistency.

Likewise, the contract `DebtToken` prevents all kinds of transfers during the "paused" state by adding the modifier to the `_clearApprovalAndTransfer` function. There is one additional operation which should likely be affected by pausing as well: `approve`. Consider adding the `whenNotPaused` modifier to the `approve` function in `DebtToken`.

Update: *Fixed in* `1ec4f35`.

Edge case in PermissionsLib leaves data in storage

The `revokeAuthorization` function in `PermissionsLib` removes the given address from the `authorizedAgents` array. The algorithm does not correctly consider the edge case when the array is reduced from length 1 to length 0. The new array length is correctly set to 0, but the address previously in the list will remain in storage, out of bounds.

Although this is unlikely to cause a problem for the semantics of the contracts, consider zeroing out the last slot in the array before decrementing its length, similarly to what `NonFungibleToken` does.

In `getExpectedRepaymentValue` of `SimpleInterestTermsContract`, `var` is used in the destructuring of a tuple.

It is encouraged to declare types explicitly, to avoid surprises with regards to integer sizes and their overflow semantics. In Solidity, `var` allows types to be deduced from the values on the right hand side of an assignment, and its use is discouraged.

To perform the same destructuring but with explicit types, first declare the variables `principalPlusInterest` and `termLengthInBlocks` with their explicit types (`uint128`), and then use those variables in a destructuring assignment without `var`: `(principalPlusInterest, termLengthInBlocks) = unpackParameters(parameters)`.

Update: *All uses of var were removed.*

Duplicate declaration of data structures

There are two `Issuance` structs, one defined in `DebtKernel` and another one in `DebtRegistry`, which are basically the same, except for one member. This duplication could cause problems if the two data structure definitions ever get out of sync during development. Consider having one definition in a central place, together with the relevant operations defined as functions.

Update: *Fixed in 0566271.*

Notes & Additional Information

- In the description of Debt Issuance Commitments in the whitepaper, the `termsContractParameters` field is documented as a `string`, inconsistently with elsewhere (including the implementation) where it is a `bytes32` value.
- `DebtToken` has three unused state variables: `brokeredTokenId`, `tokenBrokeragePermissions`, `tokenExchangePermissions`. Consider removing them.
  Update: *The variables have been removed.*

- There is two different names used to refer to the same concept: `issuanceHash` and `agreementId`. Consider consolidating to a single name.
- ERC721 is a moving target, and by now the interface implemented by the `NonFungibleToken` used is obsolete. For example, `implementsERC721` is now an optional function, and the functions `getApproved` and `transferFrom` (which is necessary for the correct functioning of `TokenTransferProxy`) are no longer part of the specification. Please keep this in mind for further development.
- There is no documentation in `DebtKernel` or in the whitepaper of the order in which the different cryptographic signatures used in `fillDebtOrder` must be sent. Consider documenting it.
- There is a limit on the number of arguments to a Solidity function which was apparently met in `fillDebtOrder`, which forced the interface to receive arrays of the different types of values. A yet experimental feature of the Solidity compiler called `ABIEncoderV2` will allow declaring external functions that receive structs as parameters. We would suggest to use this feature once it's available.
- `SimpleInterestTermsContract` and `NFTTermsContract` are abstract contracts because they don't implement all of the functions declared in the parent `TermsContract` interface. This part of the system is likely still in development.

Conclusion

One critical severity and one high severity issues were found and explained, along with recommendations on how to fix them. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Dharma contracts. We have not reviewed the related Dharma project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits