Chainlink WBTC Factory Audit

Smart Contract Security Assessment

07.10.2021

ABSTRACT

Dedaub was commissioned to perform a security audit on an augmented version of WBTC's Factory.sol smart contract on branch feature/add-chainlink-feed-check at commit hash 7ff43df67250661d2b940ccf877c1010e06798f2. The additional code can be found in this pull request. The audit also examined any other functionality highly related to Factory.sol contract. Specifically, contracts Controller.sol and Members.sol. Two auditors worked over the codebase for one day.

SETTING & CAVEATS

WBTC is an ERC20 token accompanied by peripheral smart contracts that enable the transferring of Bitcoin tokens between Bitcoin and Ethereum blockchains. Simply described, the cross-chain model relies upon the roles of "custodian" and "merchant" which form a one-to-many relation. The merchants request for transfers from the Bitcoin to the Ethereum blockchain and vice versa. For each cross-chain transfer to take place the custodian is responsible for confirming that the corresponding required transaction has taken place so as to ensure the security of the system. For example, for transferring to the Ethereum blockchain, a Bitcoin transaction transferring some amount from the merchant to the custodian is required first. On the other hand, for transferring to the Bitcoin blockchain the merchant should first burn some amount of WBTC. The custodian is trusted to perform all necessary sanity checks before minting WBTC so that no arbitrary minting takes place.

The code added by Chainlink in Factory.sol smart contract, contributes to the security of this service by enabling "Proof-of-Reserves" checks, meaning that the custodian can query a Chainlink reference feed for the total supply of Bitcoin making sure that the total amount of WBTC minted in Ethereum is not arbitrary. The values obtained from the feed are required to be up-to-date according to a configurable parameter. Both the feed address and the update parameter are configured by the custodian and can be omitted if wanted. So Proof of Reserves offers automated sanity checking with the custodian remaining fully trusted and responsible for the service's reliability.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: 01) User or system funds can be lost when third party systems misbehave. 02) DoS, under specific conditions. 03) Part of the functionality becomes unusable due to programming error.
LOW	Examples: 01) Breaking important system invariants, but without apparent consequences. 02) Buggy functionality for trusted users where a workaround exists. 03) Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed", by the client, or "resolved", per the auditors.

CRITICAL SEVERITY

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

[No low severity issues]

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend addressing them.

ID	Description	STATUS
A1	Inconsistent code style	**RESOLVED**
There are a couple of places where `block.timestamp` is used instead of the `getTimestamp()` getter. It is suggested that the latter pattern is used throughout the code, both for consistency and to reduce compilation warnings about the use of `block.timestamp`.		
A2	Why keep zero heartbeat?	**RESOLVED**
Since there is a default value for `heartbeat`, when the custodian tries to set it to zero, it is not clear why the storage variable is not instead set to the default. In the current behavior, the zero is being kept but cannot be accessed: any attempt to read it returns the default instead. It seems a bad practice to store one value but only allow a different one to be read.		
A3	Compiler known issues	**INFO**
The contracts were compiled with the Solidity compiler v0.4.24 which, at the time of writing, have some known bugs. We inspected the bugs listed for version 0.4.24 and concluded that the subject code is unaffected.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, as well as a public bug bounty program.

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.