Chainlink AToken PoR Audit

Smart Contract Security Assessment

Oct. 22 2021

ABSTRACT

Dedaub was commissioned to perform a security audit on a Chainlink extension of Aave's ATokens on branch `feature/por-atoken` at commit hash `97c11b643e9923e433c80ad3da280df2b5484e71`. The additional code can be found in this pull request. The audit also examined the extended AToken interface (`IAPoRToken.sol`), as well as the related test code (`aportoken.spec.ts`).

SETTING & CAVEATS

The audited code is of modest size, therefore the emphasis is on the interactions with other functionality. The audit emphasizes security considerations. As a standard caveat, our audit of test code is more cursory than that of smart contract code, since tests are not directly a part of the attack surface.

We note the context and limitations of the checks in the audited code. The audited functionality only ensures that the total (globally available) balances of a token used as an underlying token of an Aave AToken do not exceed the reserves of the off-chain backing token that a Chainlink feed reports. (The matching of the feed token to the "underlying" token cannot be done automatically in the code and is the responsibility of the external authorized party performing the setup.) This check of reserves is by nature very partial. If the minting of the underlying token is compromised, then using it as an underlying for ATokens is only a small part of the potential economic impact. Furthermore, the check is done only at a specific point in time: that of minting ATokens. After minting, the property can be violated, with the ATokens still available but with non-reserve-backed underlying tokens.

The developers have clarified that these shortcomings of the checking are part of the accepted assumptions.

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:

ID	Description	STATUS
L1	Emitted events do not have the intended semantics	**RESOLVED**

NewFeed and NewHeartbeat events are declared as:

```
event NewFeed(address oldFeed, address newFeed);
event NewHeartbeat(uint256 oldHeartbeat, uint256 newHeartbeat);
```

The code that emits these events is the following:

```
function _setFeed(address newFeed) external override onlyPoolAdmin returns
(uint256) {
    feed = newFeed;
    emit NewFeed(feed, newFeed);
}

function _setHeartbeat(uint256 newHeartbeat) external override onlyPoolAdmin
returns (uint256) {
    require(newHeartbeat <= MAX_AGE,
Errors.AT_POR_HEARTBEAT_GREATER_THAN_MAX_AGE);

    heartbeat = newHeartbeat == 0 ? MAX_AGE : newHeartbeat;
    emit NewHeartbeat(heartbeat, newHeartbeat);
}
```

In both cases the storage field is updated before the event gets emitted. As a result both event arguments will have the new value, instead of the first one having the old and the second one the updated value, as the event declaration suggests.

This can be easily fixed by emitting the event before the storage field gets updated.

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	External functions prefixed with underscore	**RESOLVED**

Prefixing function names with an underscore (“_”) is a practice usually followed when naming internal or private functions that are intended to be used only within the contract. However, both `_setFeed` and `_setHeartbeat` are external functions.

It is recommended that the prefix be removed from the function names.		
A2	Compiler known issues	**INFO**
The contracts were compiled with the Solidity compiler v0.6.12 which, at the time of writing, have some known bugs. We inspected the bugs listed for version 0.6.12 and believe 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.