Summary

Introducing Aave v3.1, a new opt-in upgrade of Aave V3 focused on security and operational improvements.

GitHub

GitHub - aave-dao/aave-v3-origin: Aave v3.1 protocol

Aave v3.1 protocol. Contribute to aave-dao/aave-v3-origin development by creating an account on GitHub.

Context

Aave v3 is a “living” DeFi protocol, which akin to any other software, receives upgrades over time on its different components. Sometimes, the improvements can be made in isolation in independent upgrades, for example, almost 1 year ago, we introduced a new 3.0.2 version, including only relatively minor changes and bug fixes.

However, since 3.0.2, we had multiple other ideas on the backlog heavily focused on security and optimisation, that over time got validated due to security incidents on similar protocols and by observing operational overhead for Aave contributors (e.g. governance proposals changing parameters). As some of this features in some cases were dependent between each other, end of 2023 we decided to batch them together in a new major-minor Aave v3 version we present here: Aave v3.1.

Aave v3.1

v3.1 is clearly focused in 2 fields: redundant security and optimisation of the logic to reduce operational overhead.

With those principles in mind, the following is an detailed list of the changes/improvements included into this release.

Features

1. Virtual accounting

Aave v3, is what we technically call a “dynamic” system in terms of underlying balances of ERC20 tokens. For example, to calculate utilisation and rates, the system checks how is the balance of underlying (e.g. USDC) on its aToken contract (aUSDC), and compares it with outstanding debt. Or to validate if there is enough funds for withdrawals, the system depends on not reverting on transfer() from the aToken to the user withdrawing; so also depending on ERC20 balances.

This is generally perfectly fine, but from our experience, a system like Aave should be a bit more static, meaning being less sensible to operations that don’t follow explicit interaction paths. The most classic example of this is that a donation to the aToken address of underlying should not have any effect on the stored state of the system.

The virtual accounting feature is very simple high-level: whenever there is an inflow of X capital to the protocol, that is accounted for in storage by adding X to the virtual balance variable of the specific asset; whenever there is an outflow, the opposite.

As a consequence of this, Aave has an extra layer of protection (apart from other existent ones) for different type of attack vectors used in the past of other DeFi system, following a classic security approach of defense-in-depth.

In terms of implementation, the proposal adds an optional virtualUnderlyingBalance

field for assets listed on Aave (not always used, for example on GHO), which gets modified on every action causing an inflow or outflow of capital, like on supply()

, borrow()

, withdraw()

, liquidationCall()

, etc.

The virtual balance is also now used in the interest rate strategy contract, in replacement of the aforementioned balanceOf() of underlying in the aToken. Now the formula of utilisation becomes high-level: utilisation = total debt / (virtual balance + total debt)

This new feature doesn’t create any incompatibility with Aave v3 integrations, being just additive: the asset data returned on getReserveData()

is still the same as before without virtualUnderlyingBalance

, but integrators can opt-in to use a new getReserveDataExtended()

or directly getVirtualUnderlyingBalance()

.

Given its implications and criticality, virtual accounting can be considered the major feature of Aave 3.1.

2. Stateful interest rate strategy

Having reviewed countless Aave governance rates updates, we noticed time ago that connecting new strategy contracts is a process prone to errors. To solve that, in the past we introduced an on-chain rate strategy factory, that completely removed that error-vector, deploying new rates automatically and efficiently.

However, as the rate strategy for this 3.1 needed to be changed to support virtual accounting, we decided to go a step forward and move the parameters of the rate to an storage mapping, instead of having them as immutables on separate contracts for each asset.

Implementation-wise, this feature:

- Defines a single default interest rate smart contract, to be connected to all assets listed, including those requiring a fixed rate like GHO. We kept the option to still use totally custom strategies for new listings with special dynamics on the underlying asset.
- Adds an _interestRateData

data field on the rate strategy contract, containing all the previous rate configurations for each asset (e.g. base variable borrow rate, slope1, slope2).

- Adds stricter upper limits for rates. The rationale is that on very high configured rates (e.g. hundred thousands %), predicting how the protocol reacts with specific assets' configurations becomes very chaotic. This upper limit is defined as the maximum value that base variable borrow rate + slope 1 + slope 2 can reach, and currently is configured to 1000%, a value we think it should never be reached.
- All components of the protocol connected with or depending on the rate have been updated accordingly.

3. Freezing by EMERGENCY_GUARDIAN on PoolConfigurator

Due to legacy reasons, the PoolConfigurator

didn't allow the EMERGENCY_GUARDIAN role to freeze an asset, only to pause it.

We introduced an additional contract on top (FreezingSteward) to allow this in the past, but the logic really belongs to the PoolConfigurator, so this is included into 3.1, and the FreezingSteward pattern can be deprecated.

4. Reserve data update on rate strategy and RF (Reserve Factor) changes

On Aave v3 (and v2), whenever an interest rate strategy address is replaced for an asset or the Reserve Factor changes, the reserve data is not updated (calculate liquidity/variable debt index until now and "cache" rates on the asset data).

This was simply a design choice, and even if perfectly acceptable, we decided to change it as 1) is counter-intuitive, as we think indexes until now should be updated with the old rate strategy/old RF

and 2) whenever an asset is frozen or in any state of partial functionality, the update of the rate will be factually delayed until an user makes an action.

On 3.1 we introduce logic to update reserve data whenever the rate strategy or RF of an asset changes anyhow via the PoolConfigurator

.

5. Minimum decimals for listed assets

Precision is a pretty delicate mechanism on Aave, and historically we have observed that assets with low decimals are prone to create edge case scenarios, for example, regarding inflation attacks.

Given that currently it is a pretty rare case, and usually symptom of very bad practises by the team doing the ERC20 implementation, we have introduced a validation for any asset listed on Aave to have at least 6 decimals.

6. Liquidations grace sentinel

During one security incident that required pausing the protocol we noticed that could be important to introduce a grace period for users to refill their positions or repay debt, just after the system is unpaused, for them to avoiding liquidation. This is a similar approach as with the L2 Sentinel, allowing for a grace period (in that case stopping borrowing too) whenever a downtime on a rollup network is detected.

Initially we followed the same approach as with the FreezingSteward mentioned before, and introduced on Aave v2 (the system that was affected by the pause) a LiquidationsGraceSentinel

registry/steward contract, allowing for the emergency admin to define a “delayed” pause for any asset.

However, at that point in time the mechanism was not needed on Aave v3 and slightly more complex to implement. So we postponed it until now, to make it a fully native mechanism to Aave v3.

Implementation-wise, this feature adds a gracePeriod

input parameter to pass whenever an asset is to be unpaused, which will act as a delay for liquidations.

Apart from being totally optional (it is possible to just unpause without any delay), it is heavily limited to a maximum value of 4 hours and we will recommend risk provider to always use it with maximum caution, as even if for users affected will give a window to refill collateral, it will still allow to borrow.

7. LTV0 on freezing

On previous freezing incidents, we have also noticed that when freezing an asset on v3, the correct approach, apart from halting deposits and borrows, would be to “remove” the collateral power of the asset for opening or increasing borrow positions.

For this reason, in this 3.1 we have added setting LTV to 0 atomically when freezing an asset, returning to the previous LTV value when unfreezing.

8. Permission-less movement of stable positions to variable

Consequence of a security vulnerability detected end of next year related with stable rate mode and affecting more on Aave v2, we proposed to completely deprecate it.

After getting extra approval by the community on theARFC stage, we have included on this 3.1 a function to allow permission-less movement of stable rate debt positions to variable, which factually will off-board all users borrowing at stable to variable.

Implementation-wise, this adds a swapToVariable()

function in the Aave pool, allowing any address to swap any stable rate user to variable, without changing anything else in the position.

This will only affect those v3 instances where stable rate was active at some point, so for example will not be applicable to Aave v3 Ethereum.

9. Allow setting debt ceiling whenever LT is 0

In multiple cases (e.g. stablecoins) an asset is listed as no-collateral and thus no debt ceiling.

When then at a later point the DAO decides to enable it as isolated collateral it currently can’t because of a validation checking that there are no suppliers in the pool.

The check for “no supplies” is too strict, and it was set to ensure there is no active borrows against the asset, as otherwise the ceiling account would be wrong.

The less strict, but still correct approach we added is to allow enabling of the ceiling as long as the asset is not a collateral, so validating that its Liquidation Threshold is 0.

10. New getters on Pool and PoolConfigurator for library addresses

Operationally and tooling-wise, historically has been problematic to fetch the smart contract addresses of different Solidity libraries connected to the Pool or the PoolConfigurator (e.g. PoolLogic

, BorrowLogic

, etc).

To solve that, we have added specific getters for each library on the Pool, like getPoolLogic()

or getBorrowLogic()

, returning their addresses, and opening for simple usage both on-chain and off-chain.

11. Misc minor bug fixes and sync the codebase with the current v3 production

Over time, some detected problems have received patches on production, creating certain de-sync between Github and deployed contracts, with the latest being the "head" of Aave.

With 3.1 we sync completely production and off-chain code, and in addition, we do different minor bug fixes.

Security

This 3.1 has undergone 3 different procedures:

- Security review by MixBytes.
- Security review by Certora, in addition to improved Certora properties since the v3 release.
- Security review of the most critical features (Virtual Accounting) by@stermi).

All the report will be added once finished to the Aave v3.1. repository.

The previous third-party procedures complement all the internal by BGD, that we apply to every codebase:

- Testing of all the new features, and verifying/adapting that only intended consequences appear on previous Aave v3 features.
- Gas implications.
- Carefully verifying codebase and storage diffs to avoid any type of broken behaviour.
- High-level operational implications, for example on procedures followed by other Aave contributors when doing governance proposals.

Additionally, the current Aave Immunefi bug bounty will be applicable to v3.1, once approved via governance and activated on production instances.

Licensing

Aave v3.1 follows the same licensing approach as all other major contributions of BGD Labs as service provider of the Aave DAO: BUSL 1.1 licensed to the Aave governance smart contracts, explicitly forbidden the usage of the improved codebase in production by any entity competing with Aave.

The Aave DAO has full power to change this license however deemed appropriate in the future.

The full license can be found HERE

Next steps

1. Create an ARFC Snapshot for the upgrade of all Aave v3 instances to Aave v3.1.
2. In parallel, finish the preparations of the AIP payload for the upgrade, and security procedures around it. This AIP will initialise all the new components, apart from doing the implementation upgrades themselves.
3. On-chain AIP creation.

Appendix. Aave v3.1 Origin and aave-dao

https://github.com/aave-dao/aave-v3-origin

During BGD Phase 1 and 2, almost all our developments involving smart contracts have been Foundry based. And even if generally was perfectly fine to work with it, the current hardhat-based Aave v3 repositories were causing us important friction, for example when trying to test new functionalities on updates, keeping in sync the codebase and production, or even trying to re-use different tooling we created on top of Aave v3 (e.g. stewards, config engine, custom price feeds).

Additionally, it is becoming pretty standard for security researchers to use Foundry scripts to test potential bugs or assumption of Aave, and the context switching between Solidity and Typescript (hardhat) sometimes can be very painful.

For all previous reasons, and using the opportunity of this new v3.1 version, we are releasing the Aave v3 Origin repository.

Aave v3 Origin:

- Is a repository with the whole Aave v3.1 codebase, Foundry-based.
- Hosted on an Aave DAO Github organisation, which we intend to make as the official repository for all the projects created by service providers or any type of contributors to the Aave DAO.

Proper granular permissions aligned with the role of different contributors still need to be assigned, together with moving multiple tools created by BGD (and others) for Aave; but the intention is clear: the Aave DAO should have its own Github organisation, trying to replicate as much as possible the pure decentralisation of its on-chain components

.

- The closest-to-production and fully maintained Aave codebase, trying to reduce to the maximum the confusion of having the Aave software fragmented in multiple different organisations.
- The base for official deployments of Aave, to be used by the service provider in charge of this on pre-activation stage. Aave v3 Origin unifies the previous core and periphery following a simple rationale: whenever an instance of Aave is deployed, both core and periphery are required, so there should not be separation between them.

Same as with 3.1, Aave Origin is a project licensed via BUSL 1.1 to the Aave DAO smart contracts, explicitly forbidding the usage of the codebase for production purposes by any entity competing with Aave.

The Aave DAO has full power to change this license however deemed appropriate in the future.