Summary

During the final activation phase of the Aave v3 ZKSync pool on August 21th, a problem with its behaviour was detected. Consequently, a global pause of all assets was enacted by the Aave Guardian, delaying the activation of the new instance.

The issue resulted to be not on Aave's smart contracts, but consequence of a bug in one of the dependencies of the ZKSync Era compiler: LLVM, a widely used compiler library outside the blockchain industry. This affected very specifically one logic expression of Aave, reflecting in the aforementioned unexpected behaviour.

After responsibly knowing that no production system on Era is affected, this report explains how the problem was detected, remediations executed without any funds at risk and next steps.

Before the Aave pool pausing

Below is a rough timeline of events until the Aave v3 ZKSync pool was paused:

1. app.aave.com from Aave Labs is one of the user interfaces to interact with the Aave protocol and is usually the first to provide support on new network activations.

During the voting of pool activation governance proposals (5 days), it is routine for Aave Labs to perform integration testing of both basic actions on the pool (supply, borrow, etc) and more sophisticated flows combining different operations in a pseudo-random way manner.

This procedure is complementary to the unit and integration tests of the smart contracts side performed pre-proposal stage by the proposal creator, and has two main objectives: detect any problem on the UI itself and its dependencies (e.g. helper periphery contracts aggregating data from Aave), and acting as yet another layer of redundancy of the protocol testing.

It's important to note that this testing is conducted both before and immediately after the pool's activation via governance, to eliminate any discrepancies between the fork environment and the actual production dependencies.

1. At around 15.30 GMT on 21th August, Aave Labs was performing the testing process on the UI, and detected an issue when supplying assets. This was a problem on the user interface itself, and resolved soon after.

2. Later on, another issue was detected. This problem appeared after executing a more complex flow of operations involving supplying and borrowing multiple assets, and performing a full repayment of one of the positions. The Aave Labs team observed that part of the position was disappearing from the UI.

The issue was initially narrowed down to a helper contract of the system, the UIPoolDataProvider. This contract is designed for interfaces to consume batched/processed data from the protocol. This finding was consistent with previous activations or pool upgrades, where helper contracts tended to have a larger surface area for integration issues, being non-core to the protocol.

1. After further internal research by Aave Labs, and given that the smart contracts code of Aave v3 ZKSync was same as on other networks, the main hypothesis was that the fork environment might not be reliable, as the UIPoolDataProvider seemed correct. While this research was ongoing, the pool activated and went live

2. In parallel, Aave Labs contacted BGD Labs to disclose the findings and seek deeper analysis of the potential problem.

3. Upon involvement, BGD Labs sought to answer to two immediate questions:

4. Is there really a bug?

5. If there is a bug, can it affect other Aave instances?

Almost immediately it becomes evident that there is a bug, even if it was not clear what exactly was. Concurrently, Aave Labs tested the same flow on other production networks, revealing that no other instances seemed to be affected.

1. Is there really a bug?

2. If there is a bug, can it affect other Aave instances?

3. Before conducting further research, BGD Labs decided to contact the Aave Protocol Guardian to pause the entire pool. The reasoning behind taking this step was, that with the pool being empty, there was no downside to taking the action, and theoretically, it would stop any potential attack surface.

4. The Aave Protocol Guardian demonstrated an outstanding reaction time, and the pool was paused almost immediately.

5. Following the pause of the pool, it was determined that a thorough investigation should be conducted before public communications were made. This decision was informed by suspicions that the issue might be more complex than initially apparent, and not being even totally related with Aave's contracts.

The bug research

With the pool empty and paused and no other networks affected, our priority became understanding precisely the issue.

The scenario was the following:

- The bug affected pretty deep dynamics of the pool.
- The behaviour was breaking very important high-level properties of the Aave protocol: an action on an asset (repaying) was affecting user flags related to another asset.
- Part of the pre-activation security procedures was to strictly check Solidity code differences between the Aave v3 (v3.1) contracts deployed on ZKSync and other networks' instances. The code was the same, only different in totally unrelated components with the bug in question.
- As previously commented, the misbehaviour was not present in other Aave instances, technically sharing the same Solidity code.
- The bug was also not present in ZeroLend (tested there to understand if they could be affected), a fork of Aave v3 (non v3.1) in production for months in ZKSync.

From the previous facts, our internal analysis on BGD was the following:

- Problem must be on the pool data, not on tokenization or other components, as it seemed related with user positions involving multiple assets.
- Even if we know that a repayment on an asset should not affect others, is there any place where misbehaving logic could operate on the wrong data?

From the previous analysis, we establish an initial hypothesis: somehow, the bitmap on user data containing 256 bits of boolean pairs for if an asset is borrowed and/or used as collateral by the user is getting "corrupted" by the logic, with actions disabling an asset as borrowed affecting the usage as collateral of a different asset

.

Immediately, we asked Aave Labs team members to reproduce alternative scenarios to the original that in the same environment. By our hypothesis, we thought they will show similar data corruption.

Indeed, that was the case, and a clear pattern emerged: when having multiple assets enabled as collateral and borrowed, disabling the isBorrowing flag on the bitmap more to the "left", would zeroed some of the isCollateral flags to the "right". Moreover, we generalised testing the pattern to actually prove that all flags to the "right" of the one being disabled were getting zeroed.

To better understand the scenario, this is a practical example:

- User supplies and borrows multiple assets, wstETH, USDT, USDC, which by listing order have contiguous position on the user configuration bitmap. After this, the status of the bitmap is 000...111111

(256 bits) with the 1s representing wstETH used as collateral | wstETH borrowed | USDT used as collateral | USDT borrowed | USDC used as collateral | USDC borrowed

.

- By repaying the whole debt of wstETH, the expected result would be 000...101111

, so just disabling the wstETH borrowed

boolean bit. However the result observed was 0000...100000

. Consequently, the protocol enters in an inconsistent state, where it believes the user has no borrow position, and only wstETH collateral, which is not true.

So the problem was clear, but the cause not completely. As aforementioned, the deployed contracts were exactly the same as in other networks where the issue didn't exist, exactly the same in the case of the UserConfiguration

contract dealing with bitmaps as an existing fork of Aave in ZKSync which also had no issue.

So now another hypothesis we had from the moment we got involved by Aave Labs was becoming more clear: there is a bug somewhere on custom compilation layers of ZKSync, affecting specifically the version we compiled the contracts with, but not others.

We reached out to the Matter Labs team and they started an investigation on the problem.

In parallel, we asked Certora to check deeper into the issue for detecting the root cause, while Matter Labs were performing their research . Later, by analysing the assembly output of compilation, Certora noticed the following:

- On precisely the logic section of bitmaps manipulation of UserConfiguration

, apparently some type of “magic value”/constant was introduced.

- This value was clearly not belonging to the basic bit manipulation instructions, so probably had to be a compiler optimisation: by detecting a pattern of multiple logic operations done together, it is quite common that optimisers “simplify” them with ad-hoc instructions, involving sometimes magic values/constants.

- This being a optimiser act was further proved by modifying the UserConfiguration

logic in isolation with logic doing exactly the same, but with more convoluted (and unnecessary) bit instructions. Effectively, the output of the compilation was not including anymore the “magic value”, meaning that the optimiser was simply not detecting the pattern an consequently not acting.

- More important, the “magic” value had an strange property: it had 64 bits dimension, which initially looked pretty suspicious, given that the logic was operating in 256 bits data.

This would definitely explain the bug, with a bit mask of non-expected dimension getting applied on the data, corrupting it.

This finding by Certora was communicated to Matter Labs, which in parallel and independently had reached similar conclusions. At this stage it become pretty evident that effectively it was some type of compilation bug.

The root cause of the issue detected by Matter Labs was the following.

As described on the ZKSync documentation, one of the compilation layers uses LLVM, a well-known and broadly used compiler library, applying different optimisations on the intermediate state between EVM assembly code and the final ZKSync bytecode.

The cause of the problem was actually a bug in the LLVM itself, not on the zksync side. The compiler was doing an optimisation for exactly the type of operation used on Aave, but there was a bug on the usage of signed vs unsigned constants, consequently affecting the magnitude in bits.

Important to highlight than even if that logic was buggy, compilations not doing certain optimisations could not be affected; this is why forks of Aave running on ZKSync Era were not impacted.

Considering that Aave was actually safe at all times post-pause, we have not published this report until now, after having confirmation from Matter Labs that the bug doesn’t impact any verified Era contract

.

All details on the ZKSync side can be read onhttps://x.com/zkSyncDevs/status/1829110498471621003.

Conclusion, and next steps for Aave on ZKSync

For obvious reasons, the appearance of this type of bug required a re-evaluation of the ZKSync technology from our side, in order to proceed with a new activation vote of Aave.

After internal discussion, the next steps are the following:

- Detecting an issue so deep in the tech stack is very complicated, requiring complex interaction flows that fairly frequently can only be reached by fuzzing of actions and property checking. At the same time, property checking is simply not possible at the moment, given that the tools available (e.g. Certora) are not covering custom VMs like ZKSync.

Even if the codebase has some already tests in that direction, we are improving that coverage on fork scenarios. This has some technical obstacles like compilation time of zksolc and other infrastructure limitations, but until it is performed there will be no activation proposal.

- We are continuously discussing with Matter Labs of any other potential and realistic thread on compilation stage. Compiler bugs simply exist and happened in the past on for example the Vyper compiler or solc; so even if we know that we can’t have 100% assurance of the lack of them, we want to reduce to the minimum the probability of appearance.

In line with that, we are studying reducing optimisations on middle stages to the minimum, and any other threat minimisation strategy. Exact measures are still being decided.

- Once the previous aspects are addressed, and any other that gives us full confidence, we will proceed with a new activation proposal of Aave v3 ZKSync.

In a positive note, we would like to highlight that highly frequently, bugs of this nature on the compiler layer are very difficult to detect and usually cause very high damage on anybody suffering them. However, the Aave procedures worked properly and there was pretty early detection, with absolutely zero impact.

We would like to explicitly acknowledge all the contributors involved:

- Aave Labs (@AaveLabs) for their work detecting the mis-behaviour on their user interface, and support testing our hypothesis in that environment.
- Certora (@Certora) for supporting on “low level” research once we formulated an hypothesis on BGD Labs.
- The Aave Protocol Guardian for the fast reaction on protective measures.
- The Matter Labs team for all research and remediation on the Era side, together with what we observed to be a strong security culture.

Links

- Logic on Aave being wrongly compiled https://github.com/aave-dao/aave-v3-origin/blob/main/src/core/contracts/protocol/libraries/configuration/UserConfiguration.sol#L87
- Detailed explanation of the LLVM bug by ZKSync https://x.com/zkSyncDevs/status/1829110498471621003
- Aave v3 ZKsync activation proposal https://vote.onaave.com/proposal/?proposalId=153