Strategy v2 Burn Bug Post Mortem

Alberto Cuesta Cañada

Follow

Yield Protocol

--

Listen

Share

On Friday 19th of May we were notified throughimmunefi about a bug in our Strategy contracts that would have allowed for a significant part of the funds from our LPs to be easily extracted.

A war room was convened, and it was decided to use the emergency eject function in the Strategy contracts that would immediately pull out liquidity from YieldSpace and place the contracts in a fail-safe mode where the bug could not be exploited.

Follows a description of the bug reported, the factors that allowed the bug to put funds at risk, and as well as other learnings obtained from the emergency procedures.

The Bug

The bug is in line 363 of Strategy.sol.

The function allows to burn strategy tokens to get the appropriate share of underlying pool tokens. Unfortunately, because it uses pool.balanceOf(address(this)) instead of poolCached_, the function is vulnerable to a donation of pool tokens.

The donation of pool tokens allows the attacker to obtain a bigger share of pool token reserves that he would otherwise.

After burn, the poolCached counter will only be updated by subtracting the pool tokens given from the pool reserves, so most of the pool token donation will still be in the contract, unaccounted for, and recoverable by the attacker simply by calling mint and burn again.

This process can be repeated until most of the underlying has been removed from the pool that the strategy invests in.

As part of the post-mortem, it is important to list the factors that contributed to the bug existing, the security measures that were in place, and the effectiveness of those security measures.

First factor — Coding

The bug was introduced in a contract update. This bug is not present in the v1 version of the same contract, or in the burnDivested function in the same contract. This reveals that this is not a complicated or unknown attack vector.

Subjectively, there likely was a lack of attention in coding this function. The contract was developed and tested in a very short period of time, as a simplification of the V1 contract but with the added eject

feature.

Second Factor — Review

We have coded cache-based contracts for quite some time now, and we know that they can be vulnerable to donation attacks. In reviewing YieldSpace-TV, I recall doing a search for balanceOf

through the codebase, and for each instance investigating if there was a donation based attack that would work. That's a very simple heuristic that would have avoided this mistake, at the lowest possible cost.

Third Factor — Audit

The audit failed to catch this bug. While this is technically a security measure failing to achieve the desired outcome, it is also expected that audits will miss some bugs.

Fourth Factor — Insufficient Audits

In retrospect, using a single internal audit for a non-trivial contract that would hold most of the protocol assets was insufficient. The Smart Contract Risk Assessment gave a score of 24, which required at least a contest-based audit. Cost and time-to-market pressure led to a downgrading to an internal audit, which weakened the effectiveness of this security layer.

Bug Bounty Success

The bug bounty did its job of revealing the bug before an exploit would happen. Having a high critical submission bounty cap and an active management of the bounty and submissions to achieve a steady flow of qualified security researchers poring over the code was a success.

Crisis Management and Resolution

A war room was quickly convened and worked effectively to manage the situation. The experience from previous incidents allowed us to act decisively to mitigate the risk of the bug being exploited.

However, as the first incident that affected the Strategy contracts, there are important lessons to be learned, particularly in smart contract design.

First Lesson — No Pause Mechanism

The Strategy contract has no mechanism for pausing activity and allow research to happen. This created undue risk while the results of a potential ejection action were investigated. The EmergencyBrake mechanism protecting the rest of the protocol doesn't work on Strategies or Pools. Given the code and maintenance complexity of EmergencyBrake, it is recommended that it is phased out in favour of a more traditional Pause mechanism.

EmergencyBrake is a case of premature optimization. The lack of Pause on Strategy and Pool contracts is a risk.

Second Lesson — No Upgradability

Following the example of the Sense Auto-Roller, the Strategy contract uses an eject feature to extract all assets instead of a traditional upgradability feature.

A contract upgrade would have saved considerable effort in restoring the protocol, and given that it is unclear that users strongly demand decentralization, it should preferentially be used in future contracts. The upgradability permissions can be revoked if they become controversial, maybe at the same time that an eject feature is included instead.

Third Lesson — Uneconomical Recovery Logic

The eject feature forces that all fyToken in all pools are bought at face value before unlocking the assets in the strategies. This implies a loss for the buyer, which is likely to be the protocol operators, and a profit for LPs.

In the absence of a mechanism to accurately price the fyToken, it would be appropriate to rely on a permissioned actor to replace the fyToken for an amount of underlying of his choosing.

Conclusion

The outcome of this bug was a sizable bounty paid plus an estimated two weeks of downtime for the protocol in order to restore operations. While this is not a great outcome, the bug bounty did its job and prevented a worse scenario.

Apart from the immediate bug fix, lessons were learned on emergency systems design that should reduce the impact of future instances where a bug is reported on live contracts. There are also lessons on not following adequate pre-deployment security procedures, and will influence future decisions where security is considered against cost and time to market.