zkLend security incident post-mortem

Summary

On 11th February 2025, zkLend, a money market protocol on Starknet, was attacked using an empty market exploit, causing the loss of around $9.6 million US dollars. The exploit was made against the wstETH token that was newly launched on Starknet. Initial analysis has been performed and this post-mortem serves as a brief report of the progress thus far.

Affected scope

The zkLend money market is affected, with all activities suspended.

The zkLend liquid staking protocol (kSTRK) is operating normally and unaffected by the incident.

Funds affected

The attacker extracted assets from the platform by inflating their collateral balance and borrowing funds from the system, with these transactions:

- 0x04862e266cf9a952d06a3d7e537aa68f8ba7f46d224912240b11bb9c6e7f1480: *15484.120127* USDC
- 0x0467c72d570ac97feab5ff1c2a326d1b0101c8241316a58d854c734ca7a1b446: *46410.000000* USDC
- 0x01711f45d2d6f1df2a14f7f055bdaa370b947c196dca0078b934c11a53dc3d2c: *54.240785128721525602* ETH
- 0x021120bba1e258864a64c7ab1b01784e6a9e3fc2fcd318f3b2e3b153a6454bce: *31132.071802943243729502* STRK
- 0x02a17d6753ef1dfb54d81f7ffc6eaba4b5dee6062e94519469343fe595fd5043: *81.767969838850804523* ETH
- 0x05bd9bc426b1386745afaefaf6f7bf479e00d73e491c23c31f47738e466c4648: *63.000000000000000000* ETH
- 0x07620fcae1aede36fa418dea63cdabaa81b40d2cbe7e91bcc2ff76957ce6e2a2: *90.152316563208833519* ETH
- 0x04646eff60cb19e32c56e35bac937b260d4ed26ce8b6b8a61652dfc67d02ae4b: *100.000000000000000000* ETH
- 0x0478e661d2f025f88e1ca6f6c12547aa02467fd4d4a7eb234e488028da0a7002: *190000.000000* USDT
- 0x043e316f32755736b1e005d96ec64a8a02d59a7e0d4dde82d746dcd3e726747e: *85.000000000000000000* ETH

- 0x05e88391c440b5bf3f7c700077010be36ead20c2938fc6aae006487da6ff8586: *50.000000000000000000* ETH
- 0x013596a44da5bee7ff7d2f7fee88f6c3a60dc9db8a3d3f54082c458213d374cb: *248000.000000* USDC
- 0x038e35022dfe4929c2cd9994d8a50a7e370ef866368e211cc2f4a6d602b853c6: *200000.000000* USDC
- 0x065a40f4db69f98d11f26c0a2f97a2366b4378b1851252c4a54a086d1c8e0721: *424.840062602644341198* ETH
- 0x036f7579f830345761aa0c2ee4cf5eb69a85c1e4a3008ebcf346f32c312f7d15: *116.627382140770226236* ETH
- 0x040a4805324bb7302bcb246fdb57bdde7055c17ea8d52d2a1d9aa6d9603c4b4e: *93187.048487* USDT
- 0x031920cf990baa8c809673d780e4d751aa5dc2d6f2b1b6cbd191acef436b8531: *2293067.380372347071206251* STRK
- 0x03693d14ea1a254d24b6b90a908add44bdaee8419f84cf6adb0549da1bbc7d5d: *228.138253452643699963* ETH
- 0x06e9b29d7289cc80513a77a39d3908422bcefe8ac37a025c1b7070e80e22776e: *100000.000000* USDC
- 0x0539e596de9743e0c0a8594d16d89b66a515dd38758394d73a20cfc50b5beadf: *235038.662049* USDT
- 0x00db5a4c7cc00dc0d4b9bf3542a39c863c58ee92e6b8926f451d8af77350c791: *943175.367111* USDC
- 0x02fe42f8bcdb9cf54f623a72915b111f66d1ea2bb6f4227a2f408ea672eacfe3: *286.064557341406516476* ETH
- 0x07126268ddce0d46662d3d4f08e4333a0debf416f0f9a8e76275a253d686413b: *437.457022011900399629* ETH
- 0x077c5549179d4b44aae19319f85086f403faadbf7a1db58d35b5b53022e2b9f3: *196.349965523724601445* ETH
- 0x002896d1fa68bf1a39ba8db980b4c519bc4c1d189febac58a8344b6c84e814e1: *5101830.539310479001097054* STRK

Or in aggregate:

- *2213.638314603870948591* ETH
- *1553069.487238* USDC
- *7426029.991485769316032807* STRK
- *518225.710536* USDT

which, at the time of the exploit, was worth around $9,572,151 US dollars.

Following the exploit - stolen funds were bridged from Starkent via multiple bridges including StarkGate, Orbiter, Layerswap, and rhino.fi with the following funds currently sat in the below exploiter controlled wallets:

Total unspent funds of $9,570,113 US dollars:

- 0x645c77833833a6654f7edaa977ebeabc680a9109: *$7,910,951*

- 0x0b7d061d91018aab823a755020e625ffe8b93074: *$1,138,118*
- 0xcd1c290198E12c4c1809271e683572FBF977Bb63: *$419,762*
- 0x069E70C39A6495D31f518AE2F3c1166ddEfa5cAF: *$101,282*

Initial chain tracing links the exploiter to the following exploits:

- Eralend
- OnyxDAO
- Yei Finance
- Channels Finance
- Starlay Finance

Incident timeline

All timestamps are UTC.

- *2025-02-11 12:44:35*: attacker made first contact with the zkLend market contract
- *2025-02-11 15:01:02*: attacker made first borrowing transaction using inflated collateral balance
- *2025-02-11 16:37:09* attacker made the first of a series of withdrawals from Starknet to Ethereum, Base, Arbitrum, Optimism through LayerSwap, Orbiter, and rhino.fi
- *2025-02-11 17:52:00:* zeroShadow were first made aware of the suspicious activity by Rhino.fi. Both parties agreed on their suspicion after initial check and forwarded the information to StarkWare
- *2025-02-11 19:01:00:* War room with StarkWare, zeroShadow, rhino.fi and SEAL911 was created
- *2025-02-11 19:14:00:* zkLend was made aware of an active exploit by StarkWare and zeroShadow, and subsequently joined a war room with StarkWare, the Security Alliance (Seal911), zeroShadow, and Hypernative
- *2025-02-11 19:14:57*: zkLend team made aware of suspicious activities by StarkWare security team
- *2025-02-11 19:37:36*: zkLend borrowing activities suspended following confirmation of an active exploit
- *2025-02-11 19:47:49*: with initial analysis linking to the use of flashloans, zkLend contract upgraded to disable flashloan-related functionalities
- *2025-02-11 19:50:52*: attacker initiated StarkGate transactions for bridging 1801 ETH and around 37 wstETH to Ethereum address 0x645c77833833a6654f7edaa977ebeabc680a9109
- *2025-02-11 20:09:38*: zkLend market contract further upgraded to suspend all operations
- *2025-02-12 10:08:47*: attacker withdrew *1801* ETH from StarkGate

Attack analysis

The attack happened in 2 phases:

- Phase 1: using an extremely small initial deposit, and "donations" via flash loans to cause the lending accumulator (a.k.a. index in some protocols) to grow to a large value
- Phase 2: repeatedly depositing and withdrawing to cause rounding errors, which would normally be negligible but are now significant due to the inflated lending accumulator

Accounting internals

To analyze the attack, it's important to first understand how accounting is done in the zkLend market contract.

In any lending platform, user collateral and debt amounts grow over time, by definition. What's special about blockchain-based lending protocols is that, due to the constraint on compute, the algorithm for such value accumulation must be efficient, updating many, if not all, users at a time (i.e. `O(1)`).

That's why in most lending protocols, user balances are not stored directly in face values. Instead, a raw or scaled down value is used, in combination with a global accumulator/index such that at any moment, a user's balance can be calculated as `raw balance * global accumulator`. This way, when the protocol earns revenue, the global accumulator is updated to scale all users at once.

Root cause analysis

Accumulator manipulation

There's one caveat, however: smart contracts only work with integers. Rounding errors are bound to happen, which is a well-known fact in smart contract development. This was deemed negligible, given the **assumption that `global accumulator` would be small**, which was deemed a reasonable assumption, since for example even an APY of `50%` would only yield a `global accumulator` of `1.5` after an entire year. Even in the case, which was deemed "extreme", where the `global accumulator` accumulates to `10.0`, a rounding error would only mean `10` Wei worth of difference.

Unfortunately, it turns out the actual extreme case can go *way* beyond `10.0`. When a pool is empty, the first deposit's amount is treated as baseline and `global accumulator` is initialized to `1.0`. Any subsequent revenue grows the `global accumulator`. For example, if the pool earns the same amount as the initial deposit, then `global accumulator` would grow to `2.0`.

Normally, the source of revenue is just interest from users taking out loans, which as mentioned wouldn't grow `global accumulator` much, but there's one special case: flash loans. Flash loans work like this, the market contract:

1. sends the requested amount to recipient;
2. calls a callback on the flash loan handler;
3. checks its own balance to ensure enough funds have been returned, calculated as `amount requested * (1 + fees)`.

Since flash loan users are required to return a *minimum* amount instead of an exact one, it's possible that more than needed would be returned. To handle this case, the contract **treats any increase in balance as revenue**. Effectively, this allows anyone to make "donations" to the contract.

When a new pool is created, the first depositor gets to set the baseline amount. If this amount happens to be extremely small, then it wouldn't take much "donations" to grow `global accumulator` to a rather large value, which is exactly what happened in the attack.

Withdrawal amount rounding

When handling withdrawals, the `ZToken` contract calls the `safe_decimal_math::div()` function to calculate the `raw amount` to decrease (i.e. burnt). The `div()` function behaves like what typical safe math division implementations do by default: floor division, a.k.a. rounding down.

Floor division makes sense in most cases. However, in this very specific scenario it's being used for calculating the amount of deposit certificate to burn, and rounding down means not burning as much as needed.

This alone, while technically problematic, is practically negligible. However, when combined with the accumulator manipulation issue above, it can be leveraged to create large difference between the amount withdrawn and the user balance face amount reduction, effectively allowing the attack to withdraw more than what's recorded.

Summary

To summarize, the attack was enabled by a combination of 3 separate, individually harmless/negligible design details:

1. Anyone can deposit any amount into an empty market.
2. "Donations" are accepted and settled.
3. Withdrawal raw amount rounds down.

Attack phase 1: flash loan

The attacker first made a deposit of only `1` Wei of `wstETH`, causing their raw balance to become `1` (or `0.000000000000000000000000001 wstETH`). This corresponds to the aforementioned "setting the baseline amount" part.

Then, the attacker made a "donation" of `999` Wei of `wstETH` by taking out a flash loan of `1` Wei and returning `1000` Wei. This causes the lending accumulator to move from `1.0` to `851.0`, after accounting for the treasury's share of the revenue.

The attacker made a total of 10 flash loan transactions:

- 0x039b6587b9d545cfde7c0f6646085ab0c39cc34e15c665613c30f148b569687c
- 0x0490324884ac3853facd8671f6ef57c8e853244b60c0dc978b5941128d34889b
- 0x042bf5d6b76d52236f0d962a150559dc018d8c8252fd09e03301536dfecdf3e6
- 0x01e4640b1e6b0438d7fa44350bf7c5960008e55a4078ca2730477f336975b501
- 0x05383679c03bb1830f20a84945ce84d48d60e86aed676b74500a293d910fbbeb
- 0x02625e55206406f452fdbb11e83a4990bf45da88b447e7222d854fca78390652
- 0x00f500f3a46aaadbfed0275162e0caef6da968676264c4bc8fa93961d5e129c0
- 0x021f32bfc2a18b901f4ebb2f8a027d45f6a5c9fcd8e2fff882082919a403385e
- 0x07ae78b0d15976f766aed48672ac916170fb0b5f135dbd36512c370f6ac9c902
- 0x0665f0506dbb40572693dd6111cdd0536e09af2aa114ba8afc81edc14b690271

The very fact that 10 transactions were made seems to be insignificant. The attacker's goal here was probably just to inflate the accumulator, which could have been done in one transaction if desired.

After the 10 flash loan transactions, the lending accumulator grew from the initial value of `1.0` into `4069297906051644020.0`.

Attack phase 2: deposit and withdraw

After inflating the accumulator, the attacker:

1. Deposited `4.069297906051644021 wstETH`, or just `1` Wei higher than their current calculated collateral balance. This increased the `raw balance` from `1` to `2`.

2. Deposited another `8.138595812103288042 wstETH`, or just `2` Wei higher than their current calculated collateral balance. This increased the `raw balance` from `2` to `4`.

3. Within the same transaction, withdrew `6.103946859077466029 wstETH`. The amount to burn should be calculated as:

   ```
   6103946859077466029 / 4069297906051644020.0 = 1.5
   ```

   Due to floor division, the resulting amount becomes `1` instead. The withdrawal decreased the `raw balance` from `4` to `3` only.

At this point, a discrepancy was created between the attacker's calculated collateral balance and the actual capital contributed. The contract was compromised.

The remaining part of the attacker's actions mostly involve the same routine:

1. Deposit an amount large enough to increase `raw balance` by a few units.
2. Withdraw an amount that should calculate as `1.0 < x < 2.0` (they used `1.5` the entire time) but gets rounded down to just `1`.
3. Deposit just enough to increase `raw balance` by `1` unit.
4. Repeat steps 2-3 until the amount initially put in in step 1 is mostly recovered.

Since step 3 cancels out the reduction from step 2, overall the attacker's `raw balance` increases.

Eventually, they accumulated the `raw balance` value to `1724`, representing `7015.469590033034290480 wstETH`, sufficient for draining all other pools.

zkLend audit history

zkLend's money market contracts in Cairo 0 and Cairo 1 were audited by Nethermind on 23th May 2022 (report) and 29th September 2023 (report) respectively. No critical issues were found in either report.

Subsequently, zkLend's ZEND token contract was audited on 24th November 2023 (report), while zkLend's liquid staking contracts were also audited (report) on 15th December 2024, with no critical issues found during the security review.

Response and mitigation

- **Smart contracts suspension:** The zkLend markets contract was immediately paused after the attack, suspending all deposits, withdrawals, borrowing, repayment, flash loans, and liquidations. An active warning was put out on the app's homepage.
- **Security collaboration:** Working with security experts such as zeroShadow to notify exchanges, Chainalysis, TRM and Elliptic of associated wallet addresses.
- **Fund tracking:** Continuously track stolen funds and the attacker's activities.
- **Legal collaboration:** Actively working with law enforcement (Hong Kong Police, FBI, Homeland Security) to identify and apprehend the hacker.
- **Hacker communication:** An on-chain message was sent to the hacker to seek resolution and return funds, but no response has been received.
- **Community updates:** Regular updates are being provided to users and partners regarding the protocol's status and developments.

Next steps

- **Security audit:** A security patch will be deployed and thoroughly reviewed by security teams and auditors prior to any consideration of restarting the protocol.
- **Funds recovery:** Continued work with zeroShadow to track down further fund flows.
- **User communication:** zkLend remains committed to providing regular and transparent updates to the community regarding the situation's progress and the measures being taken to secure user funds.
- **Compensation:** Gather suggestions and feedback from the community to develop a fair and appropriate compensation plan.

We extend our heartfelt thanks to our users and partners for their unwavering patience and trust during this difficult time. The security of your assets and the integrity of zkLend remain our highest priorities, and we are fully dedicated to recovering the stolen funds. Our team is tirelessly working to address every aspect of this incident, including tracking the attacker and collaborating with law enforcement and security experts. We also recognize the significant impact this breach has had on users from various protocols, including STRKfarm and Bountive. As we move forward, we will continue to provide transparent updates and take all necessary measures to protect your interests. We are also pursuing direct communication with the exploiter in hopes of a peaceful resolution while maintaining our investigation with security professionals. Your support is invaluable, and we are committed to restoring your confidence as we work diligently to resolve this issue.

We also appreciate the invaluable support provided by StarkWare, Starknet Foundation, Binance Security, Chainalysis, zeroShadow, Hypernative, Layerswap, Nethermind, Orbiter Finance, rhino.fi and SEAL911 at this time. Their assistance has been instrumental as we work diligently to address and resolve the situation surrounding the recent exploit.