FORESHOCK

Sample Report: Euler Finance

This is a sample report. It demonstrates the structure and depth of a Full Forensic Investigation deliverable using a real, fully public incident. It was not commissioned by Euler Labs or any counterparty, was not produced through Foreshock's paid engagement workflow, and is not a determination under the Forensic Engagement Terms. Every factual claim below is drawn from public sources cited in the evidence appendix; nothing here is inferred beyond what those sources state. This is research, not financial, legal, or insurance advice, and nothing below is a guarantee of accuracy for any purpose beyond illustrating report structure.

---

1. Executive summary

On March 13, 2023, an attacker exploited a missing liquidity check in Euler Finance's `donateToReserves()` function to extract approximately $197 000 000 across six lending pools on Ethereum mainnet. The attacker combined a flash loan, recursive borrowing against self-supplied collateral, a deliberate "donation" that pushed their own account underwater, and a same-block self-liquidation that let a second, attacker-controlled contract seize collateral in excess of the debt it repaid. The same pattern was repeated across the DAI, WBTC, USDC, and staked-ETH pools.

Euler Labs paused the affected module within hours, published a technical breakdown, and opened negotiations with the attacker rather than relying on recovery through law enforcement alone. Over the following three weeks the attacker returned the funds in a series of transfers, the last completing on April 3, 2023. Euler's own account of the recovery states approximately $240 000 000 was ultimately returned, more than the $197 000 000 originally taken, reflecting asset price movement between the theft and the returns. No prosecution is recorded in the public sources reviewed for this report.

This report reconstructs the incident in the structure a Foreshock Full Forensic Investigation follows: timeline, attack-path reconstruction, affected-contract analysis, transaction-level evidence, loss calculation, root-cause analysis, and remediation review, closing with a full source appendix.

---

2. Incident timeline

- **July 2022.** Euler Finance ships EIP-14, adding `donateToReserves()`, a function allowing a user to donate their own eTokens to a pool's reserves. This function is the root of the later exploit.
- **March 13, 2023, early UTC hours.** The attacker executes the exploit across multiple pools in a sequence of flash-loan-funded transactions, netting approximately $197 000 000 in DAI, WBTC, USDC, and staked-ETH derivatives.
- **March 13, 2023 (same day).** Euler Labs identifies the exploit, pauses the vulnerable module to prevent further loss, and begins publishing updates to users.
- **March 14, 2023.** Euler Labs publicly offers a $1 000 000 bounty for information leading to the attacker's arrest and the return of stolen funds, conditioned on 90 percent of funds being returned within 24 hours. Separately, Sherlock, the audit protocol covering Euler, approves and pays out a $3 300 000 claim related to the incident.
- **March 18, 2023.** The attacker returns 3 000 ETH, worth approximately $5 300 000 at the time, under 3 percent of the amount taken.
- **March 25, 2023.** The attacker returns 51 000 ETH, worth approximately $90 000 000 at the time, close to half the original loss.
- **March 27 to 28, 2023.** Further transfers return additional substantial amounts.
- **April 3, 2023, concluding at approximately 22:59 UTC.** The final transfers complete. Euler Labs' own account states the total returned reached approximately $240 000 000.
- **Later in 2023.** Euler Labs rebuilds and relaunches as Euler v2, a modular redesign, after a reported 31 separate audits before launch.

---

3. Attack-path reconstruction

The attack ran the same core pattern against each of six lending pools, using two attacker-deployed contracts: a "violator" contract that would end up holding an unhealthy position, and a "liquidator" contract that would seize its collateral. In outline, for a single pool:

1. **Fund the attack.** The attacker takes a flash loan, reported at 30 000 000 DAI from Aave, to supply working capital that is repaid within the same transaction bundle.
2. **Establish a leveraged position.** The violator contract deposits a large amount of the borrowed asset into Euler as collateral and mints (borrows) a much larger amount of the corresponding debt token against it, reported at roughly 195 600 000 eTokens against an initial 20 000 000 DAI deposit. This is ordinary, permitted borrowing on its own; nothing about it alone violates Euler's health-score requirement, reported in post-incident analyses at a 93 to 95 percent maximum loan-to-value ratio.
3. **Partially repay, then re-borrow.** The violator repays a portion of the debt (reported at 10 000 000 DAI) and immediately re-borrows a similar amount, a step that post-incident analyses attribute to manipulating the specific collateral-to-debt ratios the next step depends on.
4. **Donate collateral to the pool's reserves.** The violator calls `donateToReserves()`, reported at roughly 100 000 000 eTokens, transferring its own collateral balance into the pool's general reserves. Because `donateToReserves()` did not call Euler's own

`checkLiquidity()` health check after this transfer, the violator's account was left severely undercollateralized, with a health score reported to have fallen from a healthy value to roughly 0.75, without the transaction reverting the way a genuinely unsafe action should.

5. **Self-liquidate at the maximum discount.** With the violator now undercollateralized, the liquidator contract, controlled by the same attacker, immediately liquidates it. Because Euler's liquidation discount scaled with how far underwater a position was, and the violator was pushed deliberately far underwater, the liquidation executed at close to the maximum reported discount, around 20 percent, letting the liquidator extract collateral worth more than the debt it repaid.
6. **Withdraw and repay the flash loan.** The liquidator withdraws the drained pool balance, and the attacker repays the original flash loan out of the proceeds, keeping the remainder as profit, reported at roughly 8 870 000 DAI of profit on the DAI-pool iteration alone.

This sequence was repeated, with pool-specific parameters, across the DAI, WBTC, USDC, and staked-ETH pools, which is why the total loss is spread across four different assets rather than concentrated in one, as detailed by asset in Section 6. The pattern did not require any pool-specific exploit logic: because the missing health check lived in the shared `donateToReserves()` function every eToken contract inherited, the same six-step sequence worked against each pool by simply substituting that pool's own collateral asset, borrow amounts, and eToken balances into the same six steps. That reuse is itself part of the evidence for the root cause identified in Section 7: a defect in one shared function, not six independent, pool-specific vulnerabilities.

Why the attack needed two separate contracts rather than one. A single contract could, in principle, have donated its own collateral and then attempted to liquidate itself, but Euler's liquidation function was designed to transfer seized collateral to the caller, meaning a self-liquidating single contract would have gained nothing beyond what it already held. Splitting the roles across a violator contract (which becomes undercollateralized) and a separate liquidator contract (which performs the liquidation and receives the discount) is what actually lets the attacker extract net value from the sequence; it is not incidental structuring, it is the mechanism the profit depends on.

---

4. Affected-contract analysis

- **Euler eToken contracts (multiple pools).** The interest-bearing token contracts representing supplied collateral in each affected pool. Each pool's eToken contract exposed the vulnerable `donateToReserves()` function; the vulnerability existed identically in every pool that had adopted the EIP-14 donation feature, which is the specific, checkable reason the same pattern worked across DAI, WBTC, USDC, and staked-ETH without needing pool-specific exploit logic.
- `donateToReserves()`**.** The specific function at fault. Its purpose, letting a user voluntarily strengthen a pool's reserves, was legitimate; the defect was narrow and specific: it moved collateral out of the caller's account without then verifying the caller's account was still solvent, the same `checkLiquidity()` check nearly every other balance-affecting function on the protocol already performed.
- **Attacker "violator" contract.** A contract deployed by the attacker to hold the intentionally-undercollateralized position. Its role in the exploit is definitional, not incidental: without a distinct account willing to become unhealthy, there is nothing for the liquidator step to act on.
- **Attacker "liquidator" contract.** A second attacker-deployed contract that performed the self-liquidation against the violator's account, extracting the excess collateral. Using a separate, attacker-controlled address for this step, rather than the violator liquidating itself, matches how Euler's liquidation function was designed to work between two distinct parties, and appears in the post-incident write-ups as a deliberate choice rather than a technical necessity of the exploit itself.
- **Aave lending pool.** Not itself defective. It supplied the flash loan that funded the attack's working capital; flash loans are a standard, permissionless feature and their use here reflects the attacker's need for upfront capital, not a flaw in Aave.

---

5. Transaction-level evidence

Attacker address: `0xb66cd966670d962c227b3eaba30a872dbfb995db`, referred to in public write-ups as "Euler Finance Exploiter."

Representative attack transaction: `0xc310a0affe2169d1f6feec1c63dbc7f7c62a887fa48795d327d4d2da2d6b111d`, reported as the DAI-pool iteration of the exploit sequence described in Section 3. This is one of a larger set of transactions the attacker executed across the six affected pools; a complete, block-by-block transaction list is available by inspecting the attacker address's full history on a block explorer, which this report does not attempt to reproduce in full.

Post-exploit fund movement: a portion of proceeds was reported moved through `0xc66dfa84bc1b93df194bd964a41282da65d73c9a`, described in public write-ups as a pass-through address associated with Tornado Cash, a mixing service, before the negotiated return process described in Section 2 recovered the funds regardless.

A caveat that applies to this whole section, stated plainly rather than left implicit: the addresses and transaction hash above were compiled from secondary technical write-ups (see the evidence appendix), not independently re-derived from a block explorer for this sample report. They are reported consistently across the sources cited. A genuine paid

engagement independently confirms every address and hash directly against a block explorer and the raw transaction receipts before it is written into a delivered report; a reader of this sample should do the same before relying on any address or hash printed above for any purpose.

What a paid engagement's evidence section additionally contains, that this sample does not. Per Foreshock's published Protocol Forensics Methodology, a real engagement's automated pipeline independently parses the submitted transaction data directly from the chain, not from a secondary write-up, and a qualified reviewer separately re-verifies that parse, including independently re-deriving the loss figure from the same raw transaction data, before anything is delivered. None of that independent, on-chain parsing and re-verification was run for this sample: it was compiled from the secondary sources cited above, precisely because it is a demonstration built to show report structure, not a commissioned engagement with a submitted transaction hash to independently verify.

---

6. Loss calculation

Public sources report the following approximate amounts stolen, broken out by asset:

- **Staked-ETH derivatives:** approximately 86 000 tokens, valued at approximately $134 600 000 at the time.
- **USDC:** approximately 34 000 000, valued at approximately $34 000 000.
- **WBTC:** approximately 849 tokens, valued at approximately $18 600 000 at the time.
- **DAI:** approximately 8 900 000, valued at approximately $8 900 000.

These figures, drawn primarily from rekt.news's incident write-up, sum to approximately $196 100 000, consistent with the widely reported headline figure of approximately $197 000 000; the small difference reflects normal rounding and minor timing differences in the USD price used for each asset across sources. A separate technical analysis (CertiK) reports very close but not identical per-asset figures (846.4 WBTC, 8 877 507 DAI, 34 224 863 USDC, 73 821 stETH plus 8 080 WETH counted separately rather than combined into one ETH-derivatives figure), which this report discloses rather than silently reconciling, since the small variance between independent trackers is itself a fact worth being honest about, not a discrepancy to paper over.

Recovery. Per Euler Labs' own account of the recovery, the total amount returned reached approximately $240 000 000, which exceeds the amount originally taken because the returned assets (largely ETH) appreciated in value between the theft and the final transfer on April 3, 2023. A genuinely commissioned Foreshock engagement states both the loss-at-time-of-incident figure and any later-recovered figure separately and explicitly, exactly as done here, rather than netting them into one number that would obscure what actually happened.

How Foreshock derives a loss figure in a real engagement. The figures above are quoted as reported by the sources cited. Per Foreshock's published Protocol Forensics Methodology, a commissioned engagement instead derives the loss independently: the system's automated pipeline produces a first-pass calculation directly from the parsed on-chain transaction data, and a qualified reviewer separately re-derives that same figure

from the raw transaction data rather than trusting the system's arithmetic, with the delivered report reflecting the reviewer's own finding if it differs from the system's draft. That independent re-derivation did not happen for this sample, since it exists to show report structure, not to replace a real engagement's review step.

---

7. Root-cause analysis

The exploit traces to one specific, narrow defect: `donateToReserves()` changed a user's collateral balance without then calling the same `checkLiquidity()` health check that essentially every other balance-changing function on the protocol already called. This was not a broad architectural failure of Euler's lending design; the borrowing, collateralization, and liquidation mechanics that the attack exploited in step 5 and step 6 (Section 3) worked exactly as designed and were not themselves defective. The defect was the absence of one guard clause on one function, introduced when that function was added months before the exploit.

Two design choices made the specific defect exploitable at the scale it was:

- **Liquidation discounts that scaled with how undercollateralized a position was**, up to a reported maximum of approximately 20 percent, meant that an attacker who could deliberately and severely undercollateralize their own position (via the missing-check donation) could then extract a much larger liquidation bonus than a naturally-occurring unhealthy position would ever produce.
- **High maximum loan-to-value ratios**, reported at 93 to 95 percent, meant relatively little of an attacker's own capital was needed to establish a large enough position for the missing check to matter at scale, which is a large part of why a $30 000 000 flash loan could be turned into a $197 000 000 loss.

Neither of those two design choices is unusual or reckless on its own; both are common, deliberate trade-offs in DeFi lending design. The exploit required the missing health check specifically; without it, neither the liquidation-discount curve nor the LTV ratio would have been reachable in a way that mattered.

---

8. Remediation review

Immediate response. Euler Labs paused the affected module on the day of the exploit, limiting the loss to the pools already hit rather than allowing the same pattern to continue against any pool not yet touched.

Fix. Public post-incident analyses report the fix as adding the same `checkLiquidity()` health check to `donateToReserves()` that nearly every other balance-affecting function already had, closing the specific gap described in Section 7, alongside more comprehensive invariant testing intended to catch a function that can move balances without also verifying account health.

Recovery approach. Rather than relying solely on law enforcement, Euler Labs paired a public bounty offer (Section 2) with direct negotiation. The bounty's own stated terms, a $1 000 000 reward conditioned on 90 percent of funds being returned within 24 hours, were not met on that literal schedule: the first returned funds did not arrive until March 18,

five days after the offer, and the process ran three weeks in total rather than one day. Stated plainly rather than smoothed over, the bounty as originally structured did not work as designed; what ultimately recovered the funds was continued negotiation beyond the bounty's own deadline, not the bounty's stated terms being satisfied. Public sources reviewed for this report do not record law enforcement involvement or any prosecution; the resolution was a negotiated return of funds.

Longer-term response. Euler relaunched as Euler v2, a modular redesign of the protocol, after a reported 31 separate audits, a substantially larger audit investment than is typical before a DeFi protocol launch, consistent with a deliberate response to the March 2023 incident rather than a routine release cycle.

What this means for how Foreshock scores a protocol with this history. Under Foreshock's actual methodology, a protocol's own historical incidents carry forward permanently in its risk score, independent of how the incident was ultimately resolved or how large or established the protocol has since become; recovering the funds does not remove the incident from the record a future assessment is built on. See Foreshock's published Methodology for exactly how that carries into a live score.

---

9. How this incident feeds Euler V1's live Foreshock score today

This is not part of the forensic reconstruction above; it is included to show how a single, real, dated incident like this one actually reaches a live product number, using Foreshock's own current, published assessment rather than a hypothetical.

As of this writing, Foreshock's live assessment of Euler V1 places it in the Moderate band, with an overall score of 49.3 out of 100 under data sufficiency "sufficient." The published rationale for that score names this exact incident as one of its key drivers: historical incidents contribute 12.8 of the 100 points, described in the rationale as reflecting "1 prior incident of its own, hard floor applied regardless of current size/age." That single sentence is the practical consequence of the permanence rule described above: this March 2023 incident, now more than two years in the past and fully resolved through the recovery described in Section 2, still floors Euler V1's historical-incidents category today, and will continue to for as long as Foreshock scores this protocol.

The same published rationale also names audit history as a second driver, 8.0 of the 100 points, noting 4 audits on record and that the most recent found zero High or Medium severity findings, explicitly framed as a statement about that audit's own scope and date, not a claim that Euler V1's current code is free of issues. Read together, the two drivers illustrate the same discipline this report has tried to model throughout: a permanent, unerasable record of what happened, stated plainly, next to a current, dated, scope-limited statement of what the evidence shows now, with neither one substituting for the other.

---

10. Limitations of this sample

Stated plainly and in one place, rather than left scattered across the sections above:

- This report was compiled entirely from public secondary sources, not from independent, transaction-by-transaction on-chain parsing. Section 5 and Section 6 each disclose this where it matters most.
- No submitted policy text was evaluated against this incident, since none exists for a sample built from a public event rather than a real client engagement. The Forensic Engagement Terms describe how a real engagement checks evidence against a specific policy's coverage clauses; that step has no equivalent here.
- The transaction hash and addresses in Section 5 are reported consistently across the sources cited, but were not independently re-derived from a block explorer for this document. Treat them as a starting point for independent verification, not as a substitute for it.
- Per-asset loss figures vary slightly between the independent sources cited in Section 6. Both figures are disclosed rather than one being silently preferred, but neither should be read as a Foreshock-verified number in the sense Section 5 and Section 6 describe a real engagement producing.
- This document was not reviewed under Foreshock's Protocol Forensics Methodology's independent-reviewer step, since that step exists to check a system-generated draft against raw on-chain data for a real, paying engagement, and no such draft or engagement exists here.

---

11. Evidence appendix and sources

Every factual claim in this sample report traces to one or more of the following public sources. Where sources reported slightly different figures for the same fact, both are disclosed above rather than one being silently preferred.

- rekt.news, "Euler REKT," incident write-up covering the attacker address, transaction hash, per-asset loss breakdown, and exploit mechanism: `https://rekt.news/euler-rekt`
- Euler Finance (official blog), "War and Peace: Behind the Scenes of Euler's $240M Exploit Recovery," the primary source for the recovery timeline, bounty terms, and final recovered total: `https://www.euler.finance/blog/war-peace-behind-the-scenes-of-eulers-240m-exploit-recovery`
- CertiK, "Euler Finance Incident Analysis," independent technical analysis and per-asset figures used as a cross-check in Section 6: `https://www.certik.com/resources/blog/4iSrYY6HoaYxk1aKyjFb5v-euler-finance-incident-analysis`
- Cyfrin, "Deep Dive Exploit Analysis: Euler Finance," technical breakdown of the attack sequence used in Section 3: `https://www.cyfrin.io/blog/how-did-the-euler-finance-hack-happen-hack-analysis`
- The Block, "Euler returns to launch v2 modular DeFi lending protocol following 31 audits post-$197 million hack," source for the Euler v2 relaunch and audit count in Section 8: h`ttps://www.theblock.co/post/314655/euler-launches-v2-modular-defi-lending-protocol-following-hack`

- Foreshock's own incident ledger records this event under incident ID `defillama:Euler V1:1678665600`, sourced from DeFiLlama and rekt.news, with `amount_lost_usd: 197000000` and `amount_returned_usd: 240000000`, consistent with every figure in this report.

On methodology. This sample report was written from the public sources above, cross-checked against each other where more than one source covered the same fact, and against Foreshock's own already-published incident-ledger entry for this event. It was not produced through Foreshock's forensic pipeline (the automated draft-plus-independent-review process described in the Protocol Forensics Methodology), since there is no paying client and no submitted policy text to evaluate a claim against; it exists to show a prospective client the depth and structure of a genuine deliverable, not to replicate one end to end.