Orchard Security Bug

CONFIDENTIAL UNTIL EMBARGO LIFTED (if you're reading this and you're not Taylor, the embargo has lifted.)

Abstract

Using my `zcash-full-stack-auditor` agent framework with the recently-released Opus 4.8, I found a missing-constraint bug in the Orchard circuit which enabled undetectable and unbounded (aside from the turnstiles) inflation of ZEC within the Orchard pool.

Timeline:

(All dates/time are MDT)

Date / Time	Event
Tue 2022-05-31, 11:50 am	Vulnerable NU5 network upgrade activates
Thu 2026-05-28, ~12:00 pm	Opus 4.8 released
Fri 2026-05-29, ~6:00 pm	Issue discovered (noticed later that evening)
Fri 2026-05-29, 11:53 pm	Issue reported
Sat 2026-05-30, 6:30 am	Issue acknowledged received by ZODL
Mon 2026-06-01, ~9:30 pm	Soft fork mitigation activated
Tue 2026-06-02, ~10:00pm	Orchard re-enabled at block 3,364,600

Total window of exposure: 4 years, 1 day, 10 hours
Opus 4.8 window of exposure: 4 days, ~10 hours
Response time (acknowledged → soft fork): 2 days, 15 hours

- On Thursday, May 28, using my `zcash-full-stack-auditor` suite of agents, I began initializing an automated audit of the halo2 implementation for soundness and zero-knowledge security issues. The orchard circuit was included within the scope of this audit. The initialization phase enumerates all of the relevant code locations, spec statements, security properties, and failure modes.
- On Friday, May 29, the initialization completed. At around 2pm, I kicked off the audit phase. The audit phase assigns the appropriate type of auditing agent to each item in the checklists generated by the initialization phase. This audit was run using the recently-released Opus 4.8 model, set to max effort.

- At around 6pm Friday, an audit agent found a critical vulnerability in the Orchard circuit, claiming it could be used to double-spend Orchard notes. Some time later that evening, I noticed the finding.
- I used Claude to write a proof of concept against a circuit similar to the "diversified address integrity" part of the Orchard Action statement. This convinced me the vulnerability was likely real, and I reported it and delivered the PoC over Signal to Daira Emma Hopwood and Kris Nuttycombe 11:53pm Friday.
- To prove the concept further, I had Claude write a proof of concept against the real Orchard circuit, demonstrating that it was possible to double-spend orchard notes by revealing multiple unique nullifiers for the same note, and delivered the second PoC to the Signal group at 2:06 AM Saturday. At this time I recommended putting out an emergency release to disable Orchard.
- Through the early morning, I (along with Claude) developed an RPC test to prove the concept fully; the RPC test doubles the value of an Orchard note until the wallet's regtest Orchard balance exceeds 10M ZEC. This proved that the inflation attack was exploitable, since regtest applies all the same rules and zero-knowledge proof validation, for Orchard, as mainnet does. Regtest is a local testing mode built into zcashd; these transactions were not broadcast to testnet nor mainnet. It took ~6 hours to develop the RPC test, with Claude Opus 4.8's help. Claude did not need much guidance from me, aside from a couple hints about how to deal with action bundles being re-ordered and the multiple passes the prover does when generating proofs.
- With no response from ZODL so far (I wasn't sure if they were intentionally being silent), I called Daira Emma at 6:23am Saturday, explained that I had a full RPC test proof of concept for the issue, and directed hir towards my Signal messages. ZODL immediately began responding.
- Later on Saturday, I provided Daira Emma with the zcash-full-stack-auditor agents and gave hir a walkthrough of how it works. I also let the automated audit continue running, which turned up no new critical findings. I provided ZODL with the issues found by the audit (all Info-level, aside from already-known issues). I also ran a similar new audit on groth16/Sapling, which turned up no new critical findings.
- Sunday through Monday, I helped ZODL review the fixes. ZODL communicated the plan was to release a soft-fork disabling orchard; since I now had access to their security fixes repo, I began reviewing and testing any PRs that looked like they were on the critical path to deployment.
- ZODL prepared the soft fork release, and I ran it to observe. The first soft fork failed due to a missed p2p DoS banning rule that prematurely banned nodes unaware of the soft fork. ZODL prepared a new release with a soft fork height 60 blocks later, which activated successfully (after some contention).
- After I observed the soft fork activating, and that funds were now protected, I went to sleep early Tuesday morning.

- On Tuesday I helped review the NU6.2 release of zcashd and zebra which re-activates Orchard after the fix.
- End of updates so far Tuesday 2026-06-02 8:00pm.

Mitigating factors:

- I had done prior automated audits of the Orchard circuit, using the exact same zcash-full-stack-auditor-agents, using Opus 4.7 xhigh, and those audits did not find the bug. I later confirmed Opus 4.7 xhigh was able to find the bug, but only when directed rather specifically: "Audit the variable base scalar mul gadget for any missing constraints that could lead to an inflation bug or double-spending attack against Zcash." When prompted more generically, i.e. "Audit the orchard circuit and halo2 gadgets for any bugs that could lead to inflation or double-spend attacks against Zcash", even Opus 4.8 struggles to find the bug (only finding it in 1/4 of my test runs).
    - TODO: My handful of trials are not scientific-quality data. More trials should be run to investigate the discoverability of this issue, especially with older models.
    - TODO: The zcash-full-stack-auditor prompts leave an audit trail of what they're checking, so I can do some analysis of the old audits to see if they explicitly checked for this issue, and to better understand what path the Opus 4.8 audit took to find the issue.
    - TODO: One difference between the audit that found the bug and the prior ones that did not may have been that I fed the halo2 book in to the initialization stage (instead of just the protocol spec / ZIPs as in prior audits).
- My Claude account is in Anthropic's whitelist for security researchers. All I had to do to get on that whitelist was provide the link to the forum post saying Shielded Labs hired me; it did not do any actual identity verification (they would have been able to match my name in the forum post to my credit card on file, but I am not sure if they did that), so I think anyone with a history of public security research could get on that whitelist.
- In my test runs (and even in the audit that found the issue), Opus 4.8 was extremely skeptical that it had found a real bug, thinking that the upstream code has been audited and so it must be correct, or it must be looking at a version with a planted backdoor, and required some prodding to take the issue seriously.
- The full PoC took ~6 hours to develop with AI assistance; that time could probably have been cut in half by using /fast, since the bulk of it was waiting for token generation. If the time it took me is a representative sample, it's unlikely one of the partners the soft fork patch was privately distributed to (miners, etc.) could have found and exploited the issue before the soft fork activated. Bug details were not included in that distribution, it only stated that Orchard was being disabled for security reasons.
- Orchard had pretty in-depth audits prior to activation, suggesting the bug was difficult for humans to discover.

Aggravating factors:

- The vulnerability has existed since Orchard's inception.
- Not much Zcash technical knowledge was required to get Opus 4.8 to write the exploit.
- The vulnerability can be exploited without any observable on-chain signature. The privacy provided by the zero-knowledge proof prevents detecting the attack, since exploiting the bug only involves setting private circuit inputs to the correct value, and nullifiers produced by double-spends are indistinguishable from real nullifiers.
    - We can analyze the arity of Orchard transactions (number of Orchard actions included in a transaction); an attacker exploiting the issue might do it all in a single large transaction, or we might observe patterns of duplicating-then-consolidating notes. I had Claude do some statistical analysis of the time window since Opus 4.8 was released, and the results were inconclusive due to the normal high variance in shielded activity; there was an anomaly of arity-4 transactions in the window, but this matched up with a similar anomaly a month prior, and so could be explained by behavior patterns tied to the start/end of the month. Since the vulnerability allows doubling the value of notes, not many transactions/Orchard Actions would be needed to accrue a large balance.

Vulnerability details:

The vulnerability is in the halo2 repository's variable-base scalar multiplication gadget. This gadget is what's used to enforce part of the diversified address integrity condition in the Action statement:

> (1) pk_d^old = [ivk]g_d^old

In English, that means: the internal viewing key (ivk) supplied by the prover must be the correct one for the address (pk_d, g_d) of the note being spent.

This is what constrains the prover to use the correct incoming viewing key (ivk) for the note that they are spending, which is crucial for ensuring the nullifier revealed by the Action is the spent note's real nullifier, preventing double-spends.

A bug in the implementation allowed a malicious prover to prove this statement for any pk_d, ivk, and g_d, essentially completely circumventing the check.

To exploit the vulnerability, i.e. to spend the same note multiple times with different nullifiers, the attacker:

1. Generates a new random nullifier key nk, which is not the correct nk for the note they are spending. nk is an input to the nullifier computation, so being able to spend a note with the wrong nk gives the attacker the ability to reveal a different nullifier for that note.

2. Other constraints in the circuit force the attacker to generate the ivk = Commit^ivk_rivk(Extract_P(ak^P), nk), from the nk honestly. For the wrong nk, this will give the wrong ivk for the note, and then ordinarily (1) would prevent the attack by noticing that the attacker's supplied ivk is not the correct one for the note.
3. The attacker exploits the bug to pass the pk_d^old = [ivk]g_d^old check. Ordinarily, this condition guarantees that the prover is using the correct ivk, and therefore the correct nk, for the note.
4. Since the attacker can spend the note using a wrong nk, each wrong nk they spend the note with reveals a unique nullifier, allowing them to spend the note multiple times.

Technical details:

The vulnerability is in these two lines of code:

https://github.com/zcash/halo2/blob/32a87582dfb0ad9364ef3ffe71751ceab2a502ea/halo2_gadgets/src/ecc/chip/mul/incomplete.rs#L309-L310

> region.assign_advice(|| "x_p", self.double_and_add.x_p, row + offset, || x_p)?;
> region.assign_advice(|| "y_p", self.y_p, row + offset, || y_p)?;

The function these lines of code are in implements the double-and-add algorithm to perform scalar multiplication, taking as input an arbitrary curve point in `base`, and the bits of the scalar to multiply that point by in `bits`. (In the Orchard circuit, `base` is g_d and `bits` are ivk. The scalar multiple being computed is [ivk]g_d.)

For a performance optimization, the scalar multiplication algorithm has an intermediate stage using incomplete curve point addition formulas.

(A complete addition formula is one that correctly handles cases like adding the identity point to a point, or adding a point to itself or its own negation. An incomplete addition formula correctly handles all cases except for those. Incomplete addition is safe to use here, since the first and last stages of the algorithm use complete addition, and an attacker would have to break discrete log to hit any of the cases incomplete addition cannot handle.)

The root of the problem is that the `base` is given to the incomplete addition stage using `assign_advice()`, in the two lines cited above, which assigns a witness value (private circuit input) *without actually introducing a circuit constraint requiring that value to be equal to the actual base.*

Additional constraints in the circuit (q_mul_2) ensure that the base values used across the incomplete-addition loop are all equal to each other, but nothing constrains those values to be equal to the actual base the algorithm is supposed to be using.

A malicious prover therefore has complete freedom to choose the base used by the incomplete addition loop. Given target values of pk_d, g_d, and ivk, the malicious prover can do some algebra to work out exactly what that unconstrained base needs to be set to in order to get the result of [ivk]g_d to be pk_d. (Opus 4.8 worked out the algebra entirely on its own, I did not need to provide any hints. It would have taken me a long time to work this out without AI.)

The fix is to make the `assign_advice()` calls in the first iteration of the loop `copy_advice()`, to produce a constraint requiring the first of those values to equal the correct base, which then, through the existing q_mul_2 constraint, constrains all of them to be equal to the correct base.

Long-Term Mitigation Recommendations

- Investigate the performance hit of turning all `assign_advice()` (and similar) into constraint-generating `copy_advice()` (and similar).
    - The performance hit is probably too large, so another measure would be to enumerate all of the `assign_advice()` and somehow ask a SAT solver or AI "can this value be changed to anything other than what the honest prover assigns without violating any circuit constraint?"
        - I did this using AI just now, and it turned up no new findings.
- Create a new shielded pool identical to Orchard (or Orchard+ZSAs in NU7) and update wallets to automatically move funds so that the turnstile-enforced upper bound on the amount of potentially-stolen funds decreases over time.
- Consider some sort of multisig to turn off shielded pools; if this were available it would have reduced the exposure window by days and engineer workload by a lot.
- Balance integrity violation audits should be performed immediately upon the public release of any new AI models.