j01tz Add CVE-2020-15899 History

1 contributor

152 lines (120 sloc) | 10.5 KB

CVE-2020-15899

CVSS v3.1 Severity and Metrics	
CVSS v3.1 Vector	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:U/RL:O/RC:C/CR:H/IR:H/AR:M/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H
CVSS Base Score	9.1
Impact Subscore	5.2
Exploitability Subscore	3.9
CVSS Temporal Score	8.4
CVSS Environmental Score	9.1
Modified Impact Subscore	5.9
Overall CVSS Score	**9.1 - CRITICAL**

Summary

On Saturday June 6, 2020, the Grin security team was made aware of a critical vulnerability in the proof of work verification code. Discovered by Grin core team member and Proof-of-Work (PoW) developer John Tromp, the vulnerability, had it been exploited, would have lead to solution speedups of 1000x and more.

The vulnerability was fixed in v4.0.0-rc.1 as part of the mandatory upgrade ahead of the scheduled hard fork on July 16, 2020 (HF3). This document provides details on the vulnerability, the fix, and what measures were taken to protect Grin users.

It has been verified that this vulnerability was never exploited.

An oversight in the implementation of the Cuckaroom29 proof of work led the number of nodes in a valid graph to be restricted to half, `2^28`, rather than the expected `2^29`. As is shown below, this could lead to solution speedups in the range of 1000x or more. Even as the network no longer would use the PoW algorithm in question from HF3 and onwards, it was still vulnerable to an attack, as the speedup achieved would allow an attacker to mine from the beginning of the previous hard fork (HF2) chain and make it valid according to the longest chain rule. The fix was obscured in a refactoring of the PoW codebase that was merged and released as part of the first v4.0.0 release candidate of the Grin node.

This has been assigned CVE-2020-15899.

Recommended actions

- **None.** Nodes and wallets that are running v4.0.0 or greater are not affected by this vulnerability.

Background

Disclosure

As per Grin's security process document, the project does not currently have any established bilateral disclosure agreements. In line with our security policy, no disclosure has therefore been made to third parties prior to to this publication.

A review of the Grin++ codebase led us to determine it was not vulnerable as it was derived from the correct implementation in the Cuckoo Cycle repository at https://github.com/tromp/cuckoo/blob/master/src/cuckaroom/cuckaroom.hpp .

As the consensus rules of the network requires running software that includes a fix, no additional action is required by mining pools, wallet software providers, or exchanges.

Timeline of events

- Jun 06 2020: The Grin security team is notified of the vulnerability, an encrypted chat group is created with John Tromp.
  - It is confirmed that the vulnerability has not yet been exploited.
  - A fix is ready to go once a roll-out plan is agreed.

    - Monitor nodes are set up that are configured to detect an attack on the network and send alerts on keybase and local consoles.
    - It is determined that the current Cuckarooz verifier suffers the same bug, and that this should be addressed immediately in a refactoring / code simplification PR.
- Jun 07: Cuckarooz bug fix disguised as a simplification is introduced in PR mimblewimble/grin#3343. This leaves the bug in Cuckaroom still intact in production.
- Jun 10: Emergency action plan agreed in the event of an exploit being detected in production by the monitoring nodes.
- Jun 17: Agreement on what approach to take as Plan A. The fix will be targeted for HF3. @tromp will sneak the fix into a refactoring of all verify code shortly prior to the final 4.0.0 binaries being released.
- Jun 24: Cuckaroom bug fix disguised in a larger refactoring of all verify code is introduced in PR mimblewimble/grin#3365.
- Jun 26: @antiochp and @quentinlesceller raise questions in private about the PR and its timing and are made aware that there is an underlying security related aspect behind the PR, but not the details. The PR is merged and the fix is included in v4.0.0-rc.1 which is released the same day.
- Jul 16: At block height 786,240, the network successfully upgrades to v4.0.0 and consensus rules that fix the Cuckaroom vulnerability.
- Jul 24: Grin core team notified that a fixed critical vulnerability in Grin's PoW will be disclosed imminently.
- Jul 27: Disclosure of the vulnerability through the publication of this document.

Technical details

The vulnerability resided in the implementation of Cuckaroom29, the third instance of Grin's ASIC-resistant secondary PoW. All instances belong to the Cuckoo Cycle family of PoWs, in which the puzzle is a random graph and the solution is a proper (non-self-intersecting) cycle of length 42, which we'll call a 42-cycle.

The Cuckaroom29 graphs have 2^29 edges and 2^29 nodes, which results in an expected number of 42-cycles of approximately 1/42. The implementation was provided in Pull Request #3136 in file `core/src/pow/cuckaroom.rs`. Edge endpoints are generated by computing a siphash function on the edge index, and masking the low and high 32-bit words of the 64-bit result with a nodemask.

This file was produced by modifying a copy of the corresponding file for the previous instance `core/src/pow/cuckarood.rs`. However, whereas Cuckarood29 graphs are bipartite, with the 2^29 nodes split into two groups of 2^28 nodes, Cuckaroom graphs are monopartite with a single group of 2^29 nodes. By forgetting to adjust the setting `let nodemask = self.params.edge_mask >> 1;` to the appropriate `let nodemask = self.params.edge_mask;`, the implementation effectively restricted the Cuckaroom29 graph to only 2^28 nodes. The oversight was only noticed when preparing the fourth instance, Cuckarooz29, which doubled the number of nodes to 2^30.

This effectively constitutes a different PoW puzzle `HalfCuckaroom29`.

Almost every Cuckaroom29 42-cycle is also a HalfCuckaroom29 42-cycle (identifying nodes that only differ in bit 28). The "almost" reflects the tiny chance that node identification causes an unwanted cycle self-intersection. The reverse however fails catastrophically, as a HalfCuckaroom29 graph has an expected number of about 100B (100 billion) 42-cycles.

For the 6 months during which it was active, miners have been solving HalfCuckaroom29 as if it were Cuckaroom29. This can be seen as taking a HalfCuckaroom29 graph, splitting each of its `2^28` nodes into two, and making incident edges choose randomly among the two, to end up with a Cuckaroom29 graph. They could have instead thrown away half of the `2^29` edges in the Cuckaroom29 graph, to end up with a Cuckaroom28 graph. That gives a speedup of 2x.

We now propose a solving method that should achieve much larger speedups, of well over 1000x. It builds an efficiently indexable representation of the graph in which each node has room to store up to 4 incoming edges and up to 4 outgoing edges. Then it picks a single node `v` that has at least 4 incoming and 4 outgoing edges. It can be shown that the expected number of 42-cycles going through `v` is at least `2^14`. From there it builds, layer by layer, a tree of incoming paths each of length 21, and a tree of outgoing paths each of length 21. Denote the set of nodes found to have 21-paths to `v` as `U`, and the nodes reached by 21-path from `v` as `W`. With the average in- an out-degree equal to 2, we expect `U` and `W` to be of size at least `2^21`, which is `1/128` of all nodes. We then expect U and W to intersect in at least `2^21/128 = 2^14` nodes. These common nodes can be efficiently enumerated as can their paths to and from `v`, each forming a 42-cycle. These must still be filtered for self-intersection, but the vast majority will survive. The total effort to generate these roughly `2^14` 42-cycles through `v` is proportional to size of the double trees, or about `2^24`. If none of the 42-cycles meets the network difficulty, then the process can be repeated with a different `v`.

Fix necessity

It might appear at first, that with Cuckaroom29 no longer being active after HF3, the bug would no longer be exploitable, and would not need to be fixed. But the vulnerability would remain. An attacker, equipped with a vastly more efficient HalfCuckaroom29 solver, could begin to rewrite history all the way from HardFork2, setting initial timestamps just 1 second apart in order to drive up difficulty, afterwards setting them 60 seconds apart to stabilize difficulty, all the time while only mining the secondary PoW. The secondary scale would then be adjusted downward making secondary blocks more difficult, but would soon reach its minimum value of 13. Once the rewritten history would reach the height of HF3, its cumulative difficulty would be far higher than the real chain and a 6-month deep re-org would occur.

Fix details

To obscure the fixed nodemask setting in Cuckaroom29, a large refactoring of the PoW code was undertaken which, amongst various other changes, adds a `node_mask` field to the `CuckooParams` struct and sets it in the constructor from an additional `node_bits` parameter:

```
pub fn new(edge_bits: u8, node_bits: u8, proof_size: usize) -> Result<CuckooParams, Error> {
    let num_edges = 1u64 << edge_bits;
    let edge_mask = num_edges - 1;
    let num_nodes = 1u64 << node_bits;
    let node_mask = num_nodes - 1;
```

For the actual fix, the Cuckaroom29 struct is created as `let params = CuckooParams::new(edge_bits, edge_bits, proof_size)?;`, which passes `edge_bits=29` for parameter `node_bits`.

Links

- https://forum.grin.mw/t/next-pow-cuckaroom-unveiled-at-grincon1/6605

- https://forum.grin.mw/t/introducing-the-final-tweak-cuckarooz/7283
- https://forum.grin.mw/t/grin-v4-0-0-network-upgrade-hard-fork-3-july-2020/7001
- https://github.com/tromp/cuckoo/blob/master/src/cuckaroom/cuckaroom.hpp
- mimblewimble/grin#3136
- mimblewimble/grin#3343
- mimblewimble/grin#3365
- https://github.com/mimblewimble/grin/releases/tag/v4.0.0-rc.1