CVE-2020-6638

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	7.5
Impact Subscore	3.6
Exploitability Subscore	3.9
CVSS Temporal Score	6.5
CVSS Environmental Score	6.5
Modified Impact Subscore	3.6
Overall CVSS Score	**6.5 - MEDIUM**

Summary

On Tuesday Oct 8, 2019 a question in the dev/crypto gitter channel led to a vulnerability in the consensus protocol being discovered by the Grin developers. CVE-2020-6638 was reserved for this. The vulnerability was fixed in v3.0.0 as part of the required upgrade for the scheduled hard fork on Jan 15, 2020. This document provides details on the vulnerability, the fix, and what measures were taken to protect Grin users.

No action required by users. Nodes and wallets that are running v3.0.0+ are not affected by this vulnerability.

The vulnerability was discovered following a question on Tuesday Oct 8, 2019 in the dev/crypto gitter channel. A malicious node could introduce *block malleability* by crafting two otherwise identical versions of a valid block, differing only in the exact set of outputs being spent. This could have been used to force a chain split on the network, which in turn could have led to subsequent attacks on users being caught on the wrong side of the network split. **Such attacks would have been exceptionally difficult to pull off in practice. There has been no indication or evidence of this vulnerability ever having been exploited.**

Background

Third party 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. We have however determined that a few projects were vulnerable and have contacted them immediately upon publication.

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

- Oct 08 2019: Question relating to "UTXO vs. TXO MMRs" posted to `grin/crypto` gitter channel. Various private internal conversations discussing ramifications of this question took place, leading to an internal agreement that there is indeed a vulnerability. It's still unclear exactly how far-reaching or what precisely is impacted. Author of initial question contacted privately to request no further discussion takes place in public pending further investigation.
- Oct 10 2019: Rough plan for fix decided on. Decision made to fix "quietly" and target scheduled HF2 (Jan 2020).
- Oct 15 2019: Initial version of fix tested internally.
- Oct 29 2019: Initial draft of "enable faster sync" RFC submitted as a PR to the `mimblewimble/grin-rfcs` github repo.
- Nov 04 2019: Initial draft of PR for non-consensus breaking changes published to `mimblewimble/grin` github repo.
- Nov 26 2019: PR for non-consensus breaking changes merged to `master` branch on `mimblewimble/grin` github repo.
- Nov 29 2019: Initial draft of height specific consensus rule changes PR published to `mimblewimble/grin` github repo.
- Dec 05 2019: PR for height specific consensus rule changes merged to `master` branch on `mimblewimble/grin` github repo.
- Dec 06 2019: `v3.0.0-beta1` tagged and released. Release announcement posted on forum.

- Dec 22 2019: Grin `testnet` aka `floonet` successful hardfork incorporating "height specific consensus rule changes" fixing the vulnerability. Grin begins to commit to the UTXO set in each block header from height `290080` on `floonet`.
- Jan 03 2020: `v3.0.0` tagged and released. Release announcement posted on forum.
- Jan 16 2020: At block height 524,160, the network successfully upgraded to v3.0.0, introducing consensus rules that fix the vulnerability.
- Jan 20 2020: Disclosure of vulnerability through the publication of this document.

Technical details of CVE-2020-6638

Transaction outputs in Mimblewimble are represented as Pedersen Commitments -

$C = vH + rG$

Here v is the amount, r is the blinding factor, and G and H are two generator points on the elliptic curve.

Note that Pedersen Commitments are additively homomorphic. The sum of two commitments is equal to the commitment of the sum of the values, blinded by the sum of the blinding factors.

$C_1 + C_2$
$= (v_1H + r_1G) + (v_2H + r_2G)$
$= (v_1 + v_2)H + (r_1 + r_2)G$

Pedersen Commitments are a core primitive in Mimblewimble, keeping the transaction amounts confidential while allowing transaction outputs to be summed, verifying the integrity of the overall system. Each transaction can be shown to preserve value (nothing destroyed, nothing created) and the same can be shown across a block and across the full chain. Each transaction has an associated transaction kernel that consists of an excess value, itself a Pedersen Commitment and a Schnorr signature showing the excess is a commitment to a zero value.

Given the additively homomorphic property of the commitment scheme it is possible to construct two sets of outputs that sum to the same value. A trivial example can be demonstrated by swapping the blinding factors on pair of outputs.

$out_1 = v_1H + r_1G$
$out_2 = v_2H + r_2G$

$out_{1'} = v_1H + r_2G$
$out_{2'} = v_2H + r_1G$

$out_1 + out_2 = (v_1H + r_1G) + (v_2H + r_2G) = (v_1 + v_2)H + (r_1 + r_2)G$
$out_{1'} + out_{2'} = (v_1H + r_2G) + (v_2H + r_1G) = (v_1 + v_2)H + (r_2 + r_1)G$

$(out_1 + out_2) = (out_{1'} + out_{2'})$

Given a transaction that spends one of these pairs, we can swap the outputs being spent for the alternate pair leaving the rest of the transaction untouched. Two conflicting versions of the same transaction can be constructed that differ only in the set of outputs being spent. All other aspects of the transaction will be identical.

This idea can be extended to blocks. Two conflicting versions of the same block can be constructed that differ only in the exact set of outputs being spent. All other aspects of the block will be identical, including the block header. This implies the set of spent outputs can be replaced *after* constructing the block header and associated proof of work. We have introduced *malleability* in the block in a way that leaves the block header and associated proof of work unchanged.

Each Grin node maintains a set of global Merkle Mountain Range (MMR) structures, one each for the outputs, associated rangeproofs and transaction kernels. Each block header commits to the merkle root of each of these MMRs. On receiving a new block the contained outputs, rangeproofs and transaction kernels are appended to the existing MMRs and merkle roots verified against those committed to in the block header. Each MMR is append-only and effectively immutable. The implementation provides "rewind" functionality to support the handling of forked blocks via "truncate and append".

This raises an interesting question: If the output MMR is immutable how do we "spend" existing outputs and what data is affected when we do so? The concept of spent/unspent is handled outside the MMR itself with unspent outputs maintained in a lightweight bitmap, each bit mapping to an unspent leaf in the full output MMR. Historical spent outputs can be pruned and removed from the output leaving the root unchanged due to the immutable characteristics of the MMR structure.

Note the output MMR effectively tracks the full set of transaction outputs (TXO) while the set of unspent transaction outputs (UTXO) are represented as a bitmap mapping to a strict subset of the prunable output (P)MMR. The PMMR is immutable and append-only (while still prunable) whereas the bitmap updates over time to reflect the current UTXO set. Historical data in the UTXO bitmap can change as old outputs are spent. This allows outputs to be maintained as immutable data, minimizing the mutable data needed to track the UTXO set.

Each block has a constant reward of 60 grin. The sum of all unspent outputs must equal the total supply which is known at each block height. We verify this through the sum of the kernel excess values where each transaction kernel is of the form 0H + rG. This in turn is verified by the individual transaction kernel signatures, with rG used as the public key. So the UTXO set is verified in aggregate as we verify the sum is correct w.r.t the sum of excess values of the transaction kernels. This verifies that no value is created or destroyed and that the supply is correct. No output can be added or removed from the UTXO set without affecting the kernel sum verification.

The critical point here is that while we cannot add or remove outputs, it is possible to swap one set of outputs for another set that sum to the same overall commitment. While this cannot affect the total UTXO sum, and value cannot be created or destroyed, it does introduce the possibility of malleability to the UTXO bitmap itself. Instead of spending one set of outputs another set are spent in their place.

The UTXO set is open to malleability which in turn opens up a possibility for malleability at the block level, while leaving the block header unchanged. Recall that each block header commits to the merkle root of the full TXO set, not the UTXO set.

This allows two conflicting versions of a block to be constructed, differing *only* in the set of outputs being spent, both sharing the same identical block header.

By broadcasting one version of the block to one subset of nodes on the network and the conflicting version to another subset of nodes on the network is it possible to introduce the conditions necessary for a chain split. Once in this state the network can be forced to split by broadcasting a subsequent block containing a spend of one of the related outputs. This output will be seen as unspent and spendable by some nodes, while being seen as already spent by other nodes.

This situation is difficult to identify in advance and difficult to recover from. An arbitrarily long period of time may elapse between the introduction of the conflicting blocks setting up the necessary conditions for the chain split and the block containing the spent/unspent output that will actually force the split.

Note that during initial "fast sync" a node will not download all historical blocks. The "txhashset" chain state is downloaded from a single peer. This includes a copy of the UTXO bitmap. The conditions necessary for a chain split can be introduced here by providing a conflicting version of the UTXO bitmap directly without requiring the download of the corresponding full block. The end result is the same chain split vulnerability.

A simple example of a potential attack is as follows. Assume the following outputs exist in the current UTXO set -

A: $v_1H + r_1G$
B: $v_2H + r_2G$
A': $v_1H + r_2G$
B': $v_2H + r_1G$

(A + B) = (A' + B')

You could then create a transaction with inputs A and B that the majority of the network accepts. However, if you send a modified version tx' with the same outputs, but with inputs A' and B' instead, anyone who accepts it can be forked off the main chain at a later time.

To cause the split, you can simply spend A' or B' on the majority chain, and when the victim sees that transaction, they will believe A' and B' are already spent, so will ban the peers and not accept the block. At this point, a network split has occurred.

If you can get a large mining pool and an exchange to fall victim to the attack, the pool will continue to mine on the minority chain allowing you to send A or B to the exchange and swap for bitcoin. Since the pool and exchange are following the minority chain, both will believe A and B are still unspent and the transaction will be seen as valid.

However, on the majority chain, A and B are already spent, so the attacker would actually be performing a double-spend, and when the exchange finally realizes and resyncs to join the majority chain, the transaction they were sent is no longer valid, and the coins no longer theirs.

The attack outlined above would be exceptionally difficult to pull off in practice. The mining pool would need sufficient graph rate to continue to successfully mine several blocks after becoming split from the rest of the network, and both the exchange and the mining pool would need to be split simultaneously. This would need to happen without them noticing that both graph rate had dropped significantly, and that their connected peers were suddenly restricted.

Details of fix

Conceptually the fix is simple. Rather than committing to the set of all outputs in each block header we commit to the set of *unspent* outputs. This approach provides additional advantages beyond fixing this vulnerability. It has been discussed before in the context of making the initial sync more robust and more flexible. With the UTXO set known it is possible to break the initial txhashset download into smaller chunks and validate each chunk individually without needing to wait for the full txhashset to be downloaded from a peer. This is currently a significant bottleneck in the latency of the initial sync process. The decision was made to roll the fix for this vulnerability into preparation work for proposed improvements to the initial fast sync process.

The specific implementation choices around exactly how this is handled in Grin is covered in detail in the Enable Faster Sync RFC and the associated PR.

Briefly, the `output_root` field in each block header now commits to the combined root of the existing output PMMR and the current state of the unspent bitmap. This allows the majority of the data to continue to be maintained in the immutable (yet prunable) output PMMR while also committing to the spent/unspent information maintained in the associated bitmap. The bitmap itself is "chunked" into 1024 bit chunks, with the chunks themselves appended to an MMR. This has several nice benefits including "*locality*" (outputs spent together are often located close together, within the same chunk) and "*freshness*" (recent outputs are spent more frequently than older outputs). So while the bitmap is mutable and the data changes over time, these changes tend to occur within a relatively small number of chunks, and mainly toward the right-hand side of the MMR structure where the peaks are smaller. Any update to the bitmap will affect the overall root but hashing operations are minimized given the behavior described above.

This was a consensus breaking change as it affected the semantics of an existing field committed to in the block header. The decision was made to deploy these changes as part of the scheduled hardfork (HF2) in January 2020. The new consensus rules went into effect at block height 524160. There is no evidence or indication that this vulnerability was ever exploited prior to the hardfork. All nodes post-hardfork are running v3.0.0+ and are not vulnerable.

Links

- https://gitter.im/grin_community/crypto?at=5d9bcc6f5173c33ca187d6cf
- https://github.com/mimblewimble/grin-rfcs/blob/master/text/0009-enable-faster-sync.md
- mimblewimble/grin-rfcs#29
- mimblewimble/grin#3108
- mimblewimble/grin#3147
- https://forum.grin.mw/t/grin-grin-wallet-3-0-0-beta-1-released/6669
- https://forum.grin.mw/t/grin-grin-wallet-3-0-0-released/6853