SECURITY AND ANALYSIS REPORT FOR SOLAR CORE

VERSION 1.00: 28 NOVEMBER 2022

BACKGROUND OVERVIEW

An introduction to Solar Core, the origin of its codebase, coding and security practices, testing strategies, the health of the public testnet and areas of concern.

PROTOCOL AND IMPLEMENTATION ANALYSIS

An analysis of Solar Core, including a historical observation of upstream ARK Core vulnerabilities and their impact on design choices for Solar Core.

CONCLUSION OF KNOWN ISSUES

A concluding summary of known issues for Solar Core outlined in the earlier analysis, as well as other planned improvements not mentioned in the analysis.

APPENDIX OF SECURITY ISSUES AFFECTING ARK CORE

A complete list of open and closed security vulnerabilities in the upstream ARK Core codebase that were discovered by the lead developer of Solar Core.

DISCLAIMER

Any discussed future developments planned for Solar Core are subject to change and may not be included in the final release. This report has been written in good faith for discussion purposes only and makes no statements or warranties, express or implied, including without limitation the implied warranties of accuracy, non-infringement, merchantability, and fitness for a particular purpose.

SOLAR

TABLE OF CONTENTS

BACKGROUND OVERVIEW 6

PREAMBLE 6
THE ORIGIN OF THE CODEBASE 6
CODING AND SECURITY PRACTICES 7
TESTING STRATEGIES 8
PUBLIC TESTING 9
AREAS OF CONCERN 9

PROTOCOL AND IMPLEMENTATION ANALYSIS 10

PREAMBLE 10
PEER-TO-PEER PROTOCOL 10
POOL 13
CONSENSUS 14
CRYPTOGRAPHY 17
BLOCKS AND TRANSACTIONS 18
PUBLIC API, WS-API AND WEBHOOK SERVERS 21
BLOCK PRODUCTION 22

CONCLUSION OF KNOWN ISSUES 25

PREAMBLE 25
BLOCK FINALITY 25
OFFLINE DELEGATES 25
THEFT OF DELEGATE PRIVATE KEYS 26
DATABASE SIZE 26
DOCUMENTATION 27
PUBLIC TEST SUITE 27
CONCLUSION 27

APPENDIX OF SECURITY ISSUES AFFECTING ARK CORE 28

OPEN ISSUES 29

DELEGATE MNEMONIC ENCRYPTION LEAKS THE PASSWORD 29

MALICIOUS RELAYS HALT THE NETWORK INDEFINITELY 29

MALICIOUS RELAYS WIPE THE BLOCKCHAIN FOR EVERYBODY 30

HTTP CONNECTIONS TO THE PUBLIC API SERVER KEPT ALIVE 30

HTTP CONNECTIONS TO THE WEBHOOK SERVER KEPT ALIVE 30

FIXED NUMBER OF POOL WORKERS 31

PROPAGATING MULTIPLE VALID BLOCKS AT THE SAME HEIGHT 31

WIPING OUT THE BALANCE OF A VICTIM VIA HTLC TRANSACTIONS 32

HIGH OFFSET VALUES IN DATABASE QUERIES 32

CLOSED OR PARTIALLY RESOLVED ISSUES 33

GENERATING NEW COINS WITH MULTI-SIGNATURE TRANSACTIONS 33

INVALID BLOCK HANDLING 33

IP ADDRESS SPOOFING 34

PRODUCING BLOCKS BY INVALID OR UNKNOWN GENERATORS 34

PRODUCING MULTIPLE BLOCKS IN A SLOT AND REWARD HIJACKING 34

ZERO AMOUNT TRANSFER TRANSACTION SPAM 35

WHOLE NETWORK ROLLBACK BY TIMESTAMP MANIPULATION 35

ADDING FAKE PEERS 36

CONFLICTING DELEGATE REGISTRATION TRANSACTIONS 36

TRANSACTIONS NEAR THE PAYLOAD LIMIT STOP THE NETWORK 37

DELAYED BLOCK PROPAGATION 38

DENIAL OF SERVICE VIA LACK OF PAGINATION ON API ENDPOINT 38

SECOND SIGNATURE TRANSACTIONS HALT THE NETWORK 39

IMPROPER HANDLING OF BLOCKS WITH INVALID TRANSACTIONS 39

RACE CONDITION CAUSES INVALID BLOCKS 40

SECOND SIGNATURE REGISTRATION MANIPULATION IN THE POOL 40

BLOCK HEADER MANIPULATION IN QUORUM CALCULATIONS 40

TRANSACTIONS IN BAD BLOCKS PURGE POOL TRANSACTIONS 41

DELEGATES FORCED TO PRODUCE EMPTY BLOCKS 41

PEER-TO-PEER ENDPOINT SPAM 42

PEER LISTS BECOMING TOO LARGE 42
LARGE PLUGIN NAMES 43
LACK OF SANITISATION FOR PEER-TO-PEER ENDPOINT REQUESTS 43
BLOCKING GENUINE REQUESTS BY EXCEEDING THE RATE LIMIT 43
MALFORMED PEER-TO-PEER MESSAGES CRASH NODES 44
EXTERNALLY ACCESSING INTERNAL PEER-TO-PEER ENDPOINTS 44
PING CONTROL FRAME BOMBARDMENT 45
EMPTY HYBI WEBSOCKET HEADERS STOP NODES 45
MULTIPLE DISCONNECTION PACKETS CAUSE HIGH CPU USAGE 46
BLOCKS CONTAINING TOO MANY TRANSACTIONS 46
PLAIN HTTP CONNECTIONS CRASH THE OPERATING SYSTEM 47
TOO MANY CONNECTIONS CRASH THE OPERATING SYSTEM 47
INVALID WEBSOCKET OPCODE HANDLING 48
UNHANDLED EMITTED PEER-TO-PEER EVENTS 48
CPU INTENSIVE PAYLOADS WITH TOO MANY KEY-VALUE PAIRS 48
LOCKING WALLET BALANCES TO PREVENT USERS TRANSACTING 49
LONG LIVED HTTP REQUESTS CRASH THE NODE 49
OVERLOADING THE PUBLIC API 50
SLOW NONCE COMPARISON WHEN GETTING POOL TRANSACTIONS 50
POOL TREE STRUCTURE CAN EXCEED MAXIMUM CALL STACK SIZE 51
NO SANITISATION FOR MARSHALLED BLOCK CODEC PAYLOADS 51
INDUCED SLOW BLOCK PROPAGATION TO FORK THE NETWORK 51
HIGH AND LOW S VALUES ACCEPTED IN ECDSA SIGNATURES 52
DELAYED COMPLETION OF PEER VERIFICATION 52
BLOCK EXCEPTIONS VULNERABLE TO PREIMAGE ATTACKS 53
HALTING THE BLOCKCHAIN WITH BLOCK SCHEMA VIOLATIONS 53
PREPENDING ZEROS IN ECDSA SIGNATURE COMPONENTS 54
POOL POISONING WITH INVALID TRANSACTIONS 54
BANDWIDTH FLOOD DUE TO HTTP GET REDIRECTION 55
VERY SLOW LARGE CUMULATIVE BLOCK DOWNLOAD PAYLOADS 55
CONSECUTIVE BIG BLOCKS EXCEED MAXIMUM PAYLOAD LIMIT 56
DER SIGNATURE MANIPULATION 56
NEGATIVE VALUES ACCEPTED IN ECDSA SIGNATURES 57

OUT OF MEMORY CRASHES DUE TO HTTP HEADER MANIPULATION 57
NO MAXIMUM PAYLOAD LIMIT FOR NEW PEER CONNECTIONS 58
INSUFFICIENT TRANSACTION ASSET VALIDATION 58
PEER LISTS CAN EXCEED THE MAXIMUM ALLOWED PAYLOAD SIZE 58
INADEQUATE RATE LIMITING ON OUTGOING SOCKETS 59
OUTGOING CONNECTIONS NOT ALWAYS DESTROYED PROPERLY 59
LOCKING POSTGRESQL BY REQUESTING VERY HIGH HEIGHTS 60
BINARY DATA PAYLOADS STOP BLOCK PRODUCTION 60
BIG PAYLOADS ON INTERNAL ENDPOINTS STOP NODE OPERATION 61
BLOCKS ACCEPTED BUT NOT PROPAGATED OUT OF SLOT 61
EVADING RATE LIMITER BANNING 62
SLOW QUERY REQUESTING BLOCKS FROM SPECIFIC DELEGATES 62
DENIAL OF SERVICE VIA REVIVER FUNCTION IN TRANSPORT CODEC 62
INCOMING CONNECTIONS NOT BANNED IF BASIC CHECKS FAIL 63
SOCKET EVENT LISTENERS NOT REATTACHED ON RECONNECTION 63
UNPUNISHED SCHEMA VIOLATION REQUESTING COMMON BLOCKS 64
INSUFFICIENT VALIDATION OF MULTI-RECIPIENT TRANSFERS 64
UNSANITISED CLIENT-SIDE DISCONNECTION PAYLOADS 65
CALL ID IN PEER-TO-PEER REQUESTS SET TO ANY VALUE 65
DUPLICATE KEYS IN PEER-TO-PEER REQUESTS TAKE DOWN A NODE 66
LONG FORM LENGTH VALUES ACCEPTED IN ECDSA SIGNATURES 66
RATE LIMIT EVASION VIA HTTP HEADER MANIPULATION 66
PEER LISTS RETURN MORE PEERS THAN ALLOWED BY SCHEMA 67
TOO HIGH TIMEOUT VALUE WHEN DOWNLOADING BLOCKS 67
MAXIMUM PAYLOAD SIZE NOT RESET IN PEER CONNECTOR 68
ADDITIONAL PROPERTIES ALLOWED IN TRANSACTIONS 68
PERMESSAGE-DEFLATE ENABLED IN PEER CLIENT CONNECTIONS 69
DENIAL OF SERVICE VIA TRANSACTION FLOODING 69
INCORRECT ENDIANNESS OF TRANSACTION BUFFER 70
HTTP CONNECTIONS TO THE PEER-TO-PEER SERVER KEPT ALIVE 70
WALLET REPOSITORY POLLUTION VIA PUBLIC API ENDPOINTS 71
WALLET REPOSITORY POLLUTION VIA PEER-TO-PEER REQUESTS 71
POOL WALLET NONCE CORRUPTION LOCKING USER BALANCES 72

BACKGROUND OVERVIEW

PREAMBLE

Solar Core is a layer-1 Delegated Proof of Stake blockchain written in TypeScript. Coin holders assign vote weight to their chosen delegates in 1:1 proportion to the number of coins they hold, and it is the responsibility of the 53 delegates with the most vote weight – known as active delegates – to produce new blocks to secure the network, earning a reward in the native currency of Solar, SXP, for doing so. The delegates in the 54th position onwards are called standby delegates.

This review analyses the Solar Core blockchain application, which powers the Solar Network mainnet, which launched on 28th March 2022.

THE ORIGIN OF THE CODEBASE

Solar Core is a fork of ARK Core v3, which is a part of the ARK Blockchain Framework released by ARK Ecosystem. Originally launched in 2017, ARK Core was initially written in JavaScript and rewritten in TypeScript a year later.

ARK Core was a product of its time and some of its design choices have not stood the test of time and are not recommended for new blockchains; for example, it lacks block finality and its use of ECDSA signatures for signing blocks, messages and transactions has resulted in numerous security vulnerabilities which were all discovered and patched by the Solar Core lead developer and shared upstream. If these attack vectors had been exploited prior to their mitigation, they would have caused severe disruption, and in some cases, coin theft and unrecoverable losses for users and exchanges. Several security vulnerabilities which have been fixed in Solar Core remain open and exploitable on the upstream ARK Core.

It is worth mentioning that the ARK Core developers acknowledge these deficiencies and are developing an entirely new blockchain platform to address its faults, rather than trying to fix their existing product. Solar will not be adopting their new platform and will instead develop and improve the existing code, since it has been battle-tested in production for multiple years and is considered stable.

While these shortcomings of ARK Core will be described in more detail within this report, the developers of Solar Core have worked with the developers of ARK Core to resolve many of the known issues. For the issues that do remain, Solar Core has successfully deployed protective and preventative mitigations for all of them, which are not present in the latest public stable version of ARK Core.

CODING AND SECURITY PRACTICES

TypeScript blockchains are somewhat niche, meaning there are far less qualified independent blockchain code auditors available than for other platforms, such as Solidity smart contracts. Nevertheless, the Solar Core developers are highly security conscious, and the lead developer has discovered and resolved more than 100 security vulnerabilities across ARK Core and other TypeScript based blockchains such as Lisk.

All code is audited by the lead developer prior to being approved, undergoing thorough red team wargaming with feedback given to the developer, who adopts the role of a blue team member to ensure all shortcomings are addressed. All Solar developers strive to anticipate, detect, and mitigate any potential attacks before any code is publicly published or any code releases are made. All code must be linted and compile without any warnings or errors before approval, and every dependency must always be pinned to a specific checksum-verifiable version to eliminate the possibility of malicious modification or breaking changes.

Any code of significant importance is always published as a Pull Request on GitHub rather than a direct commit to the repository. This is then sanity checked by at least one other developer, since, for additional safety and security, the Solar Network GitHub policy requires at least one other approving code review from another accredited developer before it can be committed to the public repository.

When approved and merged, the code is released in a new version of Solar Core which is passed to the 53 decentralised delegates for approval, firstly on testnet to ensure it works and as expected under simulated real-world conditions before being deployed on mainnet. Ultimately, the network only upgrades when these delegates approve and install the new release of Solar Core.

With so many eyes tracking the code, as well as outside community developers and enthusiasts, it is impossible for a rogue developer to introduce malicious code into Solar Core. The GitHub policy also mandates the use of two factor authentication to minimise the risk of a developer account being compromised.

Solar also has a dedicated fast-track e-mail address to report any security vulnerabilities or concerns, via security@solar.org. All security vulnerabilities are promptly triaged and addressed, with a policy of assigning all reports to the highest priority level, issuing same-day fixes where necessary and practicable.

TESTING STRATEGIES

Security testing is continuous and uses a wide variety of techniques such as fuzzing inputs to API queries and peer-to-peer requests as well as block and transaction payloads. Other tests include static analysis and dependency audits. It is expected that Solar Core 5.0 will introduce a public test suite for individuals to test the platform, and this will be linked to the Solar Core GitHub repository.

PUBLIC TESTING

Solar is particularly proud of the health of its testnet, with 53 decentralised delegates all maintaining an active and stable environment for testing new releases to find issues prior to deploying them on the mainnet. Both the testnet and the mainnet launched without the need for genesis delegates since Solar has always benefited from a very active and enthusiastic community that takes their duties seriously. This can be favourably contrasted with other Delegated Proof of Stake networks such as the upstream ARK blockchain, whose testnet equivalent is heavily centralised with more than half of their active delegates being genesis delegates controlled by their team and, unlike Solar, their mainnet often receives new Core updates without any prior public testing on their testnet equivalent.

AREAS OF CONCERN

The following are the main areas of concern which are thoroughly checked during every development and testing cycle to ensure they do not occur:

- Accepting unverified blocks or transactions on the blockchain
- Denial of service attacks
- Double spending
- Incorrect selection of active delegates
- Malformed requests or responses leading to unexpected behaviours
- Performance issues affecting block production or network synchronicity
- Private key leakage
- Regressions
- Rejecting valid blocks or transactions on the blockchain

PROTOCOL AND IMPLEMENTATION ANALYSIS

PREAMBLE

To date, the lead developer of Solar Core has discovered over 90 security issues affecting ARK Core alone. This analysis covers the ARK Core protocol and implementation, outlining the lessons learned from auditing the upstream code, and how it has shaped the protocol and implementation choices for Solar Core.

This section is based on the latest mainnet release of Solar Core at the time of the report, 4.1.3, with Git revision `0eb0fd01c3253bcb59597740089fbeb3d5445f3f`. Any comparisons with ARK Core are contemporaneous unless otherwise stated, so are based on the latest mainnet release of ARK Core at the time of the report, 3.6.1, with Git revision `1ce48e4a083ce1ef14ba28a41ac5cd4cbf156b7e`.

PEER-TO-PEER PROTOCOL

The peer-to-peer protocol handles all communication between the decentralised peers on the network, including broadcasting blocks and transactions.

Following the discovery of many vulnerabilities in older versions of ARK Core due to its use of SocketCluster and JSON messages for peer-to-peer communications (a few non-exhaustive examples can be found here, here, here, here, here, here, here, here, here and here), the upstream maintainers decided to switch to using protocol buffers over Nes on top of Hapi instead. This has significantly improved the reliability of peer-to-peer communications and minimised the footprint for peer-to-peer attacks since all requests and responses must conform to a strict serialised structure. Solar Core inherits the use of protocol buffers via Nes and Hapi for its peer-to-peer layer.

However, along with the introduction of Nes and protocol buffers in ARK Core, the maintainers removed the existing peer banning code, instead exclusively relying on `iptables` rules via Linux kernel packet filtering to prevent denial of service attacks. Solar Core diverges from this approach by reinstating peer banning, meaning any connection that violates the protocol or behaves in a potentially malicious way is not allowed to reconnect for a specified duration.

The reason for this approach is due to the following factors:

1. The use of `iptables` requires deploying firewall rules post-installation via a separate script, which can be overlooked by node operators, or may conflict, override, or remove any existing firewall rules that may be set up, potentially resulting in the user being locked out of the node completely or reducing security in other unexpected, unanticipated, and undesired ways.
2. It only works with a Linux kernel that includes the `netfilter` module, meaning ARK Core is highly vulnerable to denial of service attacks when run under any non-Linux operating system such as BSD, macOS or similar.
3. It requires superuser privileges granted to the user account running ARK Core. This can potentially lead to privilege escalation vectors.
4. Multiple denial of service security vulnerabilities in ARK Core were not prevented by `iptables` rules but would have been entirely mitigated by the previous peer banning system (for example, here and here).
5. Ironically, a security vulnerability was discovered inside the `netfilter` module that ARK Core relies upon to protect itself in early 2022.

Reintroducing peer banning means Solar Core is protected against many types of denial of service attacks out of the box on any compatible operating system, including non-Linux variants, without the need for any additional configuration steps and without needing to grant any superuser privileges to the user account.

However, it is recognised that kernel-level packet filtering via `iptables` is very performant and mitigates attacks at a lower level without ever reaching the application, therefore Solar Core also provides the upstream scripts to enable `iptables` for nodes running Linux. However, the salient point is that it is not the only line of defence for Solar Core, and unlike ARK Core, it is protected from many kinds of denial of service attacks on systems that cannot use `iptables`, or have not configured it, or that run on user accounts without superuser privileges.

ARK Core requires the peer-to-peer port to be open and publicly accessible to receive blocks in real time, otherwise it can only download batches of blocks every minute and cannot reliably produce blocks. This impedes the usability and performance of ARK Core in restrictively firewalled environments and when it is running on systems connected to networks that map IP addresses by Network Address Translation (NAT). It also prevents ARK Core functioning at all if another process is using the same TCP port that ARK Core uses for peer-to-peer communications, since the port number is fixed and cannot be changed. In a security context, ARK Core delegate nodes are exposed to potential attack vectors due to the known fixed peer-to-peer port that must always be open.

Solar Core, on the other hand, operates without issues even with the peer-to-peer port closed. This reduces the attack footprint because Solar Core still works correctly, receiving and producing blocks while not accepting any incoming peer-to-peer connections. This, in conjunction with the existing IP address whitelisting system in ARK Core, means Solar Core can make outbound connections to trusted peers only, while not accepting inbound connections, reducing the risk of eclipse attacks and any potential instability arising from malicious peers trying to connect and interact with the node. It also allows users behind restrictive firewalls or NAT environments to run Solar Core nodes without restrictions, including receiving blocks in real time and reliable block production capability.

Additionally, unlike ARK Core, users are able to change the peer-to-peer port used by Solar Core to avoid port clashes with other services running on the node or to set it to a commonly used TCP port to comply with some firewall rules.

POOL

The pool holds incoming unconfirmed transactions. During block production, transactions are fetched from the pool for inclusion in the block.

The default configuration of the pool in ARK Core permits up to 15,000 transactions in the pool at any time with a limit of 150 transactions per sending wallet, which can arrive in bundles of up to 40 transactions per incoming request. Note that this is not the same as the maximum number of transactions permitted in a block, which is considerably higher, but the maximum number of incoming unconfirmed transactions that can be inserted into the pool in a single request. These values are considered sane to prevent spam and resource depletion attacks, so have also been used in Solar Core.

ARK Core introduced dynamic fees, making transactions cheaper by allowing individual delegates to set the minimum transaction fee that they would accept for inclusion in the blocks that they produce. This is achieved by using a formula based on the size of the payload and a multiplier for the type of transaction, so, for example, delegate registrations cost more than regular transfers. This also means that transactions can be prioritised by ordering the transactions in the pool by the highest fees when picking transactions to include in blocks during periods of network congestion.

However, the implementation in ARK Core is open to abuse by delegates, since they can give preferential treatment to their own transactions, and those of their

friends, by including them in their own blocks for a fee as low as 0.00000001 ARK per transaction, which they will keep anyway as delegates retain all fees in blocks they produce, effectively allowing active ARK delegates to transact for free. Meanwhile, other users pay fees that are orders of magnitude higher. Therefore, while Solar Core adopts dynamic fees from ARK Core, the implementation has been modified to enforce the dynamic fee formula on the protocol level itself, preventing delegates changing or overriding the formula to game the system to their advantage, as Solar Core rejects blocks containing transactions with fees lower than the amount calculated by the formula. This is especially important since, unlike ARK Core, the Solar Core protocol burns 90% of all transaction fees.

Earlier incarnations of ARK Core ran all incoming transaction processing on the main thread, which led to several security vulnerabilities (for example, here, here and here) so this was changed to use 3 workers to handle incoming transactions. However, since the number of workers is fixed and not correlated to the number of available CPU cores, it has been, and still is, open to exploitation. Solar Core does not use the same approach as ARK Core, instead calculating the optimum number of workers, up to a maximum of 3, based on the available CPU cores.

CONSENSUS

Blockchain consensus is the mechanism for the peers on the network to agree about the state of the blockchain. Solar Core inherits the same consensus engine used in ARK Core, but with 53 active delegates instead of 51.

The consensus engine of ARK Core is basic. Each round consists of a fixed number of blocks equal to the number of active delegates, and each active delegate is entitled to produce a block in that round. Active delegates are those that have not resigned and have received the most votes from coin holders. This ranking is

recalculated for each new round. Each round consists of a specific pseudorandom order in which each delegate may produce its block, and each delegate has 8 seconds to produce a block when it is their turn, which is known as a slot. Failure to do so means that no block is produced in that slot. If one or more delegates fail to produce a block, once all delegates have been given an opportunity to produce a block, the order loops back to the start so the first delegate in the round is given an opportunity to produce another block, and so on, until the requisite number of blocks have been produced for the round.

As said, the consensus engine is basic. One major flaw with the ARK Core implementation is that it allows any peer to contribute to consensus calculations regarding the state of the network. This means that with enough malicious peers and cryptographic knowledge, anybody can convince legitimate peers that their blockchain state is incorrect, with catastrophic consequences such as erasing all blocks and transactions in the blockchain, or halting the blockchain entirely. The developers of Solar Core concluded that these issues were too severe and unacceptable, so deployed a simple yet highly effective mitigation solution that is missing in ARK Core. Namely, only the state of the active 53 delegates may contribute to consensus calculations regarding the state of the network, rather than any peer as is the case with ARK Core. This means that these open and unfixed attack vectors in ARK Core are not realistically executable on Solar Core.

Another issue is the lack of block finality, meaning that delegates can, either accidentally or maliciously, produce and propagate multiple valid blocks within their slot, with some peers on the network receiving a different block to the rest, or delay the propagation of their block so that only a part of the network has received their block by the time the next delegate is due to produce theirs. This means the network as a whole cannot agree about the state of the blockchain, resulting a network fork, potentially halting the blockchain (here or here). For

ARK Core, these forks can take minutes or longer to resolve, leading to prolonged downtime and instability. Solar Core has mitigated this with nearly instantaneous fork recovery, so even in the absence of block finality, the negative impact of producing and propagating multiple valid blocks is drastically minimised.

Nevertheless, as a safety precaution arising from the lack of block finality, hashed timelock contracts (HTLC) are not enabled on the mainnet, owing to the possibility of network forks resulting in transactions being reversed. It is planned to accomplish block finality in Solar Core 5.0 by replacing the basic ARK Core consensus engine with a new engine built by the Solar Core developers, where a supermajority of delegates must cryptographically sign each block within a single aggregated signature, with punishments for witnessing multiple blocks in the same slot or at the same height.

Another issue is that the selection of active delegates simply relies on picking the delegates at the start of each round that have the most vote weight and have not resigned. There are no checks to ensure that the delegate node is operational and synchronised with the network. This means a malicious user can deliberately use their votes to elect offline delegates as active delegates, resulting in network degradation and potentially network suspension, depending on the number of offline active delegates. A mitigation for this is planned in Solar Core 5.0 by selecting only the most voted delegates with nodes that are cryptographically confirmed as being online and synchronised with the network.

Additionally, a new infraction transaction type is planned for Core 5.0 which will allow a quorum of delegates to vote to punish delegates that are harming the network if the infraction can be proved and verified by blockchain data or metrics, such as failing to produce blocks in many consecutive rounds.

CRYPTOGRAPHY

Cryptography is the beating heart of any cryptocurrency, underpinning the security of the blockchain by preventing fraudulent transactions and ensuring non-repudiation on the immutable blockchain via secure digital signatures. ARK Core and Solar Core use `bcrypto` with native bindings for better performance of cryptographical operations and both use the `secp256k1` elliptic curve to implement their public key cryptography, which is the same curve used by Bitcoin.

ARK Core uses ASN.1 DER-encoded ECDSA signatures for blocks, while transactions and messages can be signed with ECDSA or legacy Bitcoin Cash Schnorr signatures. The same public keys are used for both ECDSA and legacy Bitcoin Cash Schnorr signatures as both schemes are deterministic due to their RFC6979 compliance. However, this determinism has a grave security implication that can leak the private key due to nonce reuse by, for example, tricking a user into signing a message using a version of the ARK Desktop Wallet that uses ECDSA, then getting them to sign the same message with the same private key with a later version of the ARK Desktop Wallet or ARKVault, which use legacy Bitcoin Cash Schnorr signatures. Once leaked, a hacker gains full control over the wallet.

Moreover, ASN.1 DER-encoded ECDSA signatures are problematic because, unlike Schnorr signatures, they are inherently malleable. This means it is possible to produce multiple valid signatures for any payload without any knowledge of the private key. This can be accomplished by transforming (*r, s*) to (*r, −s mod n*) or by tampering with the ASN.1 DER encoding, and has led to multiple transaction replay and network forking vulnerabilities in ARK Core (here, here, here, here and here). While all presently known ECDSA and ASN.1 malleability vectors have been patched in ARK Core, a lingering issue is that it is only possible to rule out further future ECDSA malleability vectors by relying on stronger than

usual assumptions. In addition, unlike Schnorr signatures that are always 64 bytes long, ECDSA signatures are variable length. This makes enforcing a tight schema more difficult, since it is not possible to clamp the length to a fixed size.

Solar Core addressed all these issues by removing ECDSA and ASN.1 DER encoding from the very beginning. Originally, blocks, messages and transactions were signed using the same legacy Bitcoin Cash Schnorr signatures as ARK Core, before migrating to BIP-340 Schnorr signatures which, in addition to not being malleable, are also not deterministic, protecting against nonce reuse attacks.

BLOCKS AND TRANSACTIONS

Accurate block and transaction processing ensures that wallet balances and other attributes are reliably updated when blocks and transactions are applied or reverted. It also safeguards against invalid data being added to the blockchain.

ARK Core ships with the following transaction types:

- Type 0: Single-recipient transfer
- Type 1: Second signature registration
- Type 2: Delegate registration
- Type 3: Vote
- Type 4: Multi-signature registration
- Type 5: IPFS hash
- Type 6: Multi-recipient transfer
- Type 7: Permanent delegate resignation
- Type 8: Hashed timelock contract lock
- Type 9: Hashed timelock contract claim
- Type 10: Hashed timelock contract refund

Following the resolution of most security vulnerabilities affecting block and transaction processing in the early life of ARK Core (a few non-exhaustive examples are here, here, here, here, here, here, here, here, here and here), the current release of ARK Core includes mostly robust block and transaction handling routines, although there still remains an unfixed issue involving balance theft via HTLC claim and refund transactions, and their attempt at fixing wallet repository pollution is only a partial solution since it does not prevent pollution from empty wallets when reverting blocks and transactions. Solar Core inherits the same block and transaction processing logic as ARK Core, but with these remaining issues fully resolved; the former by ensuring HTLC fees are handled correctly and the latter by taking a completely different approach to ARK Core by re-indexing wallets involved in transactions after applying or reverting them.

Solar Core has temporarily disabled multi-signature registrations as the multi-signature system will be rewritten, likely in Solar Core 5.0, to use aggregated signatures to significantly reduce transaction overhead. The upstream implementation requires the full Schnorr signature of every participant in each multi-signature transaction, which can lead to blockchain bloat.

The ARK Core vote transaction implementation allows for one delegate to be voted in one transaction only, by using the public key of the delegate. This is wasteful as the unique name of the delegate could be used instead, reducing storage requirements from a fixed 33 bytes to a variable maximum of 20 bytes. Therefore, the Solar Core implementation uses delegate usernames instead and allows for multiple delegates to be voted in a single transaction, further reducing overhead and storage requirements as users no longer need to split their balance across multiple wallets and cast individual votes when wanting to support more than one delegate. This also reduces friction, so users can more easily participate in the voting process to elect delegates which is essential for network security.

The ARK Core permanent delegate resignation type – which irrevocably rescinds the eligibility of a delegate to produce blocks, for example, when their private key has been compromised – has been made more flexible in Solar Core by introducing temporary delegate resignations which can be revoked later. This is so, for example, if a delegate node experiences a fault that cannot be quickly rectified, the delegate may temporarily resign so the highest voted standby delegate replaces it as an active delegate until the node is operational again, at which time the delegate can revoke its resignation and resume producing blocks. This provides better network resilience and operational security compared with the alternative scenario of a non-functional delegate failing to produce blocks for a prolonged period, degrading the performance of the network.

Another modification to the delegate resignation type is that it does not incur a transaction fee in Solar Core. This is to encourage the use of the resignation mechanism if a delegate is no longer able to fulfil its obligations, since the default ARK Core concept of delegate resignations being chargeable transactions disincentivises their use and opens the door for abuse if the private keys of an active delegate wallet are leaked, since a malicious user can run a script to continually drain the balance of the compromised delegate so it cannot pay for the resignation transaction. At the same time, the malicious user could spam the network by producing multiple blocks by the compromised delegate at the same height, causing network forks which could not be stopped. The fee-free approach for delegate resignations in Solar Core prevents this kind of attack.

A separate concern is that, if the private key of a wallet is stolen, the hacker gains unlimited control over it. While second signature registrations are possible in both ARK Core and Solar Core so that effectively two private keys are required to transact, if both keys are compromised via a keylogger or similar, the hacker can still access the wallet. To help protect against this, it is anticipated that Solar Core

5.0 will introduce a new transaction type to assign multiple public keys to a wallet via a permissions system to grant limited privileges to the corresponding private keys. This will provide granular access controls for the privileged keys to limit the actions they can carry out, such as restricting the types of permitted transactions or enforcing a spending limit. These privileged keys can be revoked at any time via the main private key. Keeping the main private key safely stored offline and exclusively using these restricted privileged keys to transact on a day to day basis will limit the actions an attacker could take with a compromised key.

PUBLIC API, WS-API AND WEBHOOK SERVERS

The Public API server provides a REST API and is the primary method for users to interact with the blockchain either by requesting data or posting transactions. The WS-API server features the same data and endpoints as the Public API but via WebSockets, while also adding the ability to subscribe to events so the server can push data to the client rather than the client polling the API for updates. Similarly, the webhook server allows a user to register for events and then the server will send a payload to a predefined URL when certain conditions are met.

All three servers are powered by Hapi, which is the same framework used for the peer-to-peer server. The WS-API is unique to Solar Core, but the Public API and webhook servers are inherited from ARK Core.

Regrettably, the ARK Core maintainers do not consider Public API or webhook server attacks as security vulnerabilities and, consequently, there are exploitable issues in ARK Core that have been fixed in Solar Core, such as keeping alive HTTP connections on the Public API and webhook servers, despite upstream fixing the same issue on the peer-to-peer server with the same framework, and an ongoing ability to stop nodes operating by using high offset values in database queries.

BLOCK PRODUCTION

Block production is the process of creating a new block containing unconfirmed transactions currently in the pool and submitting the new block to the blockchain.

Due to the modular nature of ARK Core, and by extension, Solar Core, the logic to produce blocks resides in a separate module, which is configured either by command line interface (CLI) parameters or by manually editing `delegates.json`, which is a file containing the private keys of all delegates configured on the node.

The block production module in ARK Core has an unresolved issue which can lead to password and private key leakage when used in conjunction with BIP38 password encryption. Although it can be argued that a plain text mnemonic or private key can be leaked by reading the `delegates.json` file, the issue with the upstream BIP38 password encryption implementation leaking the password is that, due to the common human trait of reusing passwords, leaking the password opens the door for a hacker to potentially access other accounts belonging to the user if the same password was reused elsewhere. Arguably this is more dangerous than storing the private key or mnemonic in plain text, and since the ARK Core CLI advocates the use of BIP38 encryption as the “recommended” option, it is reasonable to assume most users would follow the unsafe recommendation and use BIP38 rather than plain text.

The fact that this issue exists and remains unfixed in a blockchain application with cryptography at its heart, for more than one year after it was reported to the ARK Core maintainers, is deeply concerning. Users should rightfully expect that a cryptography-based application would, at the very least, know how to keep their encryption password secret and never leak it in plain text so trivially. This was unacceptable for Solar Core and was resolved prior to the launch of the mainnet.

Solar Core removed the insecure upstream BIP38 encryption implementation to be socially responsible by not allowing users to inadvertently expose passwords that could potentially have catastrophic consequences on their lives. It is for the same reason that, in contrast to ARK Core, Solar Core converts any plain text mnemonic in `delegates.json` to its hexadecimal private key – a one way unrecoverable process – in case the user chose a simple password as their mnemonic rather than generating it from the recommended BIP39 word list.

It is also planned for Solar Core 5.0 to include a new block production module, built from the ground up, which can securely handle BIP38 password encryption. This will also support the new consensus engine, which will require delegates to assign a new block production public key to their wallet, which is separate from any other key associated with their wallet and can be revoked and re-issued if the corresponding private key is compromised. It is the block production private key that will be stored in the new replacement for `delegates.json`. This will add much greater resilience to the network and prevent the need for delegates to resign if their block production private key is compromised.

On the topic of consensus, as mentioned earlier, there can be issues arising from propagating multiple blocks at the same height. One way this can accidentally occur is by inadvertently running multiple instances of the block production module for the same delegate on different servers. Therefore, it is of paramount importance that only one instance of the block production module is ever active for any delegate at any time, and that multiple servers must not be used to produce blocks for the same delegate. However, Solar Core has introduced some additional checks to the block production module that are not present in the upstream ARK Core block production module to attempt to catch more cases when multiple block production modules are active for a delegate and stop block production in those circumstances.

Another weakness with the upstream ARK Core block production module is that it will only attempt to produce a new block once, at the start of a slot. If, due to network latency or other factors, the delegate node did not receive the previous block by this time, it will be unable to produce a valid block. This means that even if the node regains network synchronisation and receives the missing block during the slot, it will still fail to produce a new block because it will not try again.

A slot lasts for 8 seconds, so it is wasteful to not try again throughout this window of opportunity. Therefore, the block production module in Solar Core changes the upstream behaviour to keep trying throughout the remainder of the slot if it fails. It will retry once per second, until the slot is over, or until it succeeds. This improves network security by minimising the number of missed blocks that arise from transient connectivity issues, packet loss, or higher than usual latency.

CONCLUSION OF KNOWN ISSUES

PREAMBLE

This section summarises the known issues that are present in Solar Core as mentioned in the previous analysis, and touches on a couple of other issues not mentioned in the analysis that would benefit from additional improvements.

BLOCK FINALITY

The consensus engine inherited from ARK Core does not support block finality, meaning it is theoretically possible to rewind and remove many blocks and transactions from the supposedly immutable blockchain.

However, it is extremely important to stress that Solar Core has multiple mitigations not present in the upstream ARK Core code which prevent the practical exploitation of attack vectors that can adversely affect the blockchain arising from the lack of block finality in the consensus engine. Nevertheless, a more robust consensus engine including block finality is being developed for the next major release of Solar Core which will give more peace of mind and permit the use of hashed timelock contracts on the Solar Network mainnet.

OFFLINE DELEGATES

The only criteria to become an active delegate is to gain enough votes from coin holders and to not be currently resigned. This means that, with enough votes, users could elect non-functional offline delegates that fail to produce blocks. Similarly, if a previously functioning delegate node stops working, they will also fail to produce blocks and the performance of the network will be degraded.

Planned improvements to the per-round delegate selection algorithm will help to prevent this by only selecting the most voted delegates whose nodes are online, synchronised, and responsive.

THEFT OF DELEGATE PRIVATE KEYS

If the private key of a delegate is compromised, a delegate must permanently resign to protect the integrity of the network since blocks are produced and signed using the private key of the delegate. This is unfortunate, as the delegate must then try to gain enough voter support under a new username otherwise their delegacy ambitions are over.

This should be avoidable with the future introduction of separate block production keys tied to each delegate wallet that can be revoked and re-issued by the delegate if they are leaked or stolen.

DATABASE SIZE

The PostgreSQL database size is growing at a larger than anticipated rate for the Solar Network mainnet blockchain, in part due to the design decisions of the ARK Core maintainers to include the full JSON asset payload in every transaction row. This has been compounded by the popularity of multiple-delegate voting, which can lead to large JSON asset payloads which are being stored in the database.

Code has already been published to swap out the PostgreSQL database with an embedded SQLite3 database along with significant schema refactoring to dramatically shrink the size of the blockchain database, and these changes should arrive in the next major release of Solar Core.

DOCUMENTATION

While most of the online documentation is comprehensive and accurate, some parts may be outdated as everything was initially based on the upstream documentation provided by ARK. Unfortunately, it has not yet been fully updated to accommodate the changes and improvements made to Solar Core, and this is further compounded by errata in the original upstream documentation.

Work is ongoing to fix any errors and omissions in the documentation that may currently be present.

PUBLIC TEST SUITE

The test suite from ARK Core is not compatible with Solar Core due to the plethora of changes that have been made to the code. While the Solar Core developers always rigorously test the code privately prior to deployment on the public testnet and then mainnet, it is recognised that a public test suite would boost external user confidence in the safety and robustness of Solar Core.

It is anticipated that a brand new public test suite will be included in the next major release of Solar Core, which will also be publicly linked to GitHub Actions so anybody can review and analyse the outcome of running the public test suite.

CONCLUSION

Solar Core is a secure layer-1 blockchain with many changes made to the ARK Core code to resolve all known security issues. It has a clear path forward to build a better blockchain, with a socially responsible outlook to strengthen the security of the blockchain for the benefit of all users of the platform.

APPENDIX OF SECURITY ISSUES AFFECTING ARK CORE

This appendix lists all the security vulnerabilities that have affected the upstream ARK Core which were discovered by the lead developer of Solar Core. They are included here to demonstrate the scale of the thorough in-depth audit that the lead developer of Solar Core has already undertaken for the platform inherited by Solar Core, and to highlight that the deep scale of the audit – covering every aspect from the Public API to the peer-to-peer protocol to the cryptography to the pool and everything else in between – is why the developers of Solar Core are confident about the security of the platform going forward, since they believe all existing major vulnerabilities have been discovered and addressed by virtue of a years-long continual audit of ARK Core by the lead developer of Solar Core.

It is worth emphasising that most of these issues were never live in Solar Core, as most were already resolved in either the upstream ARK Core product – exclusively because of the tireless and dogged determination of the lead developer of Solar Core – or the pre-release Solar Core code prior to the launch of the Solar Network mainnet meaning that Solar Core was never at risk of exploitation from most of these issues.

For the issues that are still live in ARK Core, it is important to emphasise that Solar Core has deployed partial or full mitigations for all of them to protect against their dangers, which are sufficient to ensure that they cannot be exploited on the network under real-world conditions.

The lead developer of Solar Core disclosed all the issues outlined in this document to the maintainers of ARK Core several months, and in some cases, years in advance of the publication of this report.

OPEN ISSUES

These issues have been fixed in the public Solar Core repository but not in ARK Core. They are in chronological order, by earliest date of resolution in Solar Core.

DELEGATE MNEMONIC ENCRYPTION LEAKS THE PASSWORD

The BIP38 encryption functionality in ARK Core is inherently unsafe because the password can be easily retrieved in plain text since it is passed to processes as a cleartext parameter, and with that, it is trivial to decrypt the private key.

Solution: Remove BIP38 mnemonic encryption.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-02-09 (`Solar-network/core@9bb4564`)

MALICIOUS RELAYS HALT THE NETWORK INDEFINITELY

For a delegate node to produce blocks on ARK Core, a quorum of at least 66% of all peered relay nodes must agree with the network state of the node. By spinning up a few hundred cheap relays with falsified state data, the quorum percentage will drop below the 66% threshold. In these conditions, ARK Core will immediately and indefinitely halt as delegates can no longer produce blocks by virtue of having insufficient quorum. While halted, users are not able to make any transactions or meaningfully access or spend their coins.

Solution: Only include elected active delegates in quorum calculations without being skewed by any relays.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-03-03 (`Solar-network/core@09cc554`)

MALICIOUS RELAYS WIPE THE BLOCKCHAIN FOR EVERYBODY

ARK Core continues to use relay nodes in consensus decisions. This highly dangerous behaviour means a malicious actor armed with sufficient knowledge can launch a swarm of relay nodes signalling that they are forked at a very early height. If this swarm represents an absolute majority of all relays, all other nodes will trigger a rollback in batches of 5,000 blocks until reaching this early height, permanently erasing millions of blocks and transactions along the way.

Solution: Only permit elected active delegates to participate in consensus decisions involving fork recovery.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-03-03 (`Solar-network/core@09cc554`)

HTTP CONNECTIONS TO THE PUBLIC API SERVER KEPT ALIVE

A user could spam the Public API server with malformed HTTP requests to generate a `400 Bad Request` response but keep the connection open to mount a flood attack, which would stop the node operating normally.

Solution: Close the connection after every request.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-05-30 (`Solar-network/core@716a897`)

HTTP CONNECTIONS TO THE WEBHOOK SERVER KEPT ALIVE

As above, a user could spam the webhook server to flood the node and stop it operating properly. Note that the webhook server is disabled by default.

Solution: Close the connection after every request.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-06-03 (`Solar-network/core@2124677`)

FIXED NUMBER OF POOL WORKERS

ARK Core always spawns 3 pool workers to handle incoming transactions. If a server has less than 3 CPU cores available, resource depletion can occur due to maxing out all available cores by flooding the pool with incoming invalid transactions. This would starve the main thread, resulting in degraded performance, loss of network synchronicity and an inability to produce blocks.

Solution: Calculate the maximum number of pool workers to spawn based on the number of available CPU cores.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-06-16 (`Solar-network/core@2843746`)

PROPAGATING MULTIPLE VALID BLOCKS AT THE SAME HEIGHT

A malicious delegate can produce multiple distinct blocks and broadcast them to different peers, so parts of the network receive different valid blocks at the same height by the same delegate, causing prolonged network instability.

Solution: Replace the consensus engine so a supermajority of delegates must agree on a valid block before adding it to the blockchain. Until then, the nearly instantaneous fork recovery in Solar Core is sufficient to neutralise the vector.

Fixed in ARK Core? No
Fixed in Solar Core? Partial on 2022-06-16 (`Solar-network/core@4877dac`)

WIPING OUT THE BALANCE OF A VICTIM VIA HTLC TRANSACTIONS

In its default configuration, ARK Core does not permit setting fees for HTLC Claim and HTLC Refund transactions (which is a weakness as a congested network relies on higher fees to include transactions in blocks). Nevertheless, any project building on ARK Core can change this, and by doing so, they would be vulnerable to a huge flaw in the processing of these fees. The implementation allows anybody to send a HTLC Claim or HTLC Refund transaction, even if they are not a party to the lock, however, the beneficiary would always pay the transaction fee, rather than sender of the transaction. A malicious user could send a HTLC Claim or HTLC Refund transaction with a huge fee that they would never pay, equal to the balance of the wallet of the victim, erasing their entire wallet balance.

Solution: Ensure the sender of the transaction always pays the fee.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-08-07 (`Solar-network/core@919ddcb`)

HIGH OFFSET VALUES IN DATABASE QUERIES

Specially crafted requests to the Public API server could cause high CPU usage and resource starvation. If parallelised, these requests could stop a node working.

Solution: Solar is not currently affected due to its small number of blocks. However, refactoring the database schema and API to not produce slow queries is a more permanent solution. Delegate node operators are strongly advised to disable the Public API or use IP address whitelisting to prevent public access.

Fixed in ARK Core? No
Fixed in Solar Core? Yes on 2022-11-17 (`Solar-network/core@9e34120`)

CLOSED OR PARTIALLY RESOLVED ISSUES

These issues have been fixed in the public repositories of both ARK Core and Solar Core. They are in chronological order, by earliest date of resolution.

GENERATING NEW COINS WITH MULTI-SIGNATURE TRANSACTIONS

In a multi-signature transaction, the transaction handler only verified the signatures and did not properly conduct balance checks. This made it possible to generate new coins on the network utilising a multi-signature transaction.

Solution: Perform wallet balance checks for multi-signature transactions.

Fixed in ARK Core? Yes on 2018-12-06 (`ArkEcosystem/core@97c3876`)
Fixed in Solar Core? Yes on 2018-12-06 (`Solar-network/core@97c3876`)

INVALID BLOCK HANDLING

The `lastDownloadedBlock` variable was not reset when discarding invalid blocks. This caused network nodes to continually attempt to download new blocks from the wrong height, effectively halting the network. This issue would have allowed a malicious user to disrupt network nodes and the network itself.

Solution: Reset `lastDownloadedBlock` after discarding an invalid block.

Fixed in ARK Core? Yes on 2018-12-10 (`ArkEcosystem/core@3d7baf9`)
Fixed in Solar Core? Yes on 2018-12-10 (`Solar-network/core@3d7baf9`)

IP ADDRESS SPOOFING

The whitelist could be bypassed by IP spoofing as ARK Core blindly trusted any value in the `X-Forwarded-For` header. This could fill the peer list with loopback IP addresses to cause a denial of service attack and prevent block propagation.

Solution: Use `request.info.remoteAddress` from the Hapi server to get the IP address of the peer rather than relying on the `X-Forwarded-For` header.

Fixed in ARK Core? Yes on 2018-12-11 (`ArkEcosystem/core@a3c70fb`)
Fixed in Solar Core? Yes on 2018-12-11 (`Solar-network/core@a3c70fb`)

PRODUCING BLOCKS BY INVALID OR UNKNOWN GENERATORS

Anyone can sign and broadcast a block regardless of whether they are an active delegate, but ARK Core would always fork when receiving a block from an unexpected generator, even if it was invalid or unknown. If a malicious actor kept broadcasting such blocks, the blockchain would stall.

Solution: Distinguish between inactive/unknown and active generators and only fork in the latter case, otherwise silently discard the block.

Fixed in ARK Core? Yes on 2019-01-14 (`ArkEcosystem/core@d30fbdd`)
Fixed in Solar Core? Yes on 2019-01-14 (`Solar-network/core@d30fbdd`)

PRODUCING MULTIPLE BLOCKS IN A SLOT AND REWARD HIJACKING

An active delegate could produce multiple blocks within the allocated 8 second slot time if the block IDs were different and with an incrementing block height (last block height + 1), as each additional block was accepted. This generated

block rewards for each extra block produced in the slot, resulting in inflated rewards per round for any delegate carrying out this exploit.

Solution: Add an additional check for the slot number of a received block, which must be greater than the slot number of the previous block.

Fixed in ARK Core? Yes on 2019-01-14 (`ArkEcosystem/core@e350826`)
Fixed in Solar Core? Yes on 2019-01-14 (`Solar-network/core@e350826`)

ZERO AMOUNT TRANSFER TRANSACTION SPAM

The transaction schema is designed to reject transfer transactions if the amount is not a positive number. This means that zero amount transfers are meant to be invalid. However, this check was not enforced at consensus-level, so an active delegate could remove that schema check so zero amount transfers would be accepted by their node to be added to a block, then all other nodes would also accept them, leading to transaction spam and a bloated blockchain.

Solution: Enforce additional schema checks at consensus-level.

Fixed in ARK Core? Yes on 2019-01-28 (`ArkEcosystem/core@f04fe92`)
Fixed in Solar Core? Yes on 2019-01-28 (`Solar-network/core@f04fe92`)

WHOLE NETWORK ROLLBACK BY TIMESTAMP MANIPULATION

A malicious delegate could craft a block at the correct height but with a timestamp from a previous round that collided with a valid slot time for that delegate in the current round. ARK Core would incorrectly accept such a block and then immediately enter fork recovery mode and roll back the blockchain. All nodes could receive this bad block simultaneously, triggering a rollback of the whole network, permanently erasing blocks and transactions.

Solution: Discard blocks with a past timestamp without triggering fork recovery.

Fixed in ARK Core? Yes on 2019-01-31 (`ArkEcosystem/core@8a62f55`)
Fixed in Solar Core? Yes on 2019-01-31 (`Solar-network/core@8a62f55`)

ADDING FAKE PEERS

Peers using non-quad-dotted notation representation could be added to peer lists, for example, `localhost` or any decimal from `2130706433` to `2147483646` which would resolve to `127.0.0.1` to `127.255.255.254`, all being valid loopback addresses in a Linux installation. This could crash a node and eventually stop the network as the poisoned peer list would propagate and infect all peers.

Solution: Ensure IPv4 addresses are converted into quad-dotted notation.

Fixed in ARK Core? Yes on 2019-01-31 (`ArkEcosystem/core@8a62f55`)
Fixed in Solar Core? Yes on 2019-01-31 (`Solar-network/core@8a62f55`)

CONFLICTING DELEGATE REGISTRATION TRANSACTIONS

A user could send one payload of two or more transactions containing delegate registrations for the same username but from different wallets, or a single transaction containing a delegate registration for a username, immediately followed by another transaction containing a delegate registration for the same username but from a different wallet, and this could be repeated numerous times prior to the first delegate registration transaction being included in a block. All these transactions would be accepted, added the pool, and broadcast to other peers to be added to their pools too. The next delegate would then fail to produce a valid block because the block would contain multiple delegate registration transactions for the same username, which would be rejected as it

would fail to apply the second delegate registration onwards since the username would be already taken by the first delegate registration, so the entire block would be rejected as invalid. As the conflicting transactions were broadcast to all peers but not included in a block, they remained in their respective pools, so the next delegate in line would attempt to include them in another block with the same outcome, which would have a cascading effect, resulting in a stalled network because no delegate would be able to produce a valid block as they would all try to include the conflicting transactions.

Solution: Add pool and protocol checks to forbid conflicting transactions.

Fixed in ARK Core? Yes on 2019-02-11 (`ArkEcosystem/core@8399caa`)
Fixed in Solar Core? Yes on 2019-02-11 (`Solar-network/core@8399caa`)

TRANSACTIONS NEAR THE PAYLOAD LIMIT STOP THE NETWORK

The maximum HTTP payload was 1MiB, but nothing ensured that blocks only contained transactions that would fit in a block below that size. Anybody could send oversized transactions that would approach (but not exceed) this limit since such oversized transactions could be easily crafted by stuffing extra data into the `signatures` field in any transaction payload to pad it up to the 1MiB limit. Although nodes would accept these transactions as valid, when it was time to include them in a block, the additional block headers would make the block size larger than 1MiB. All nodes would reject the block with HTTP error `413 Request Entity Too Large` and the oversized transactions would remain in the pool, stopping the production of blocks until the oversized transactions expired.

Solution: Add pool and protocol checks to prevent oversized transactions.

Fixed in ARK Core? Yes on 2019-02-13 (`ArkEcosystem/core@7691cef`)
Fixed in Solar Core? Yes on 2019-02-13 (`Solar-network/core@7691cef`)

DELAYED BLOCK PROPAGATION

If a malicious delegate produced a block but delayed broadcasting it to the next delegate, the next delegate in the round would produce a block at the wrong height as it would not have received the previous block. By the time the victim delegate attempted to broadcast their own block at the same height, the delayed block would have been received by most or all the network, meaning the newly produced block by the victim at the same height would be rejected.

Solution: Use the height from the network rather than the local node when producing a block.

Fixed in ARK Core? Yes on 2019-03-12 (`ArkEcosystem/core@eb6b092`)
Fixed in Solar Core? Yes on 2019-03-12 (`Solar-network/core@eb6b092`)

DENIAL OF SERVICE VIA LACK OF PAGINATION ON API ENDPOINT

The `/api/delegates/{delegate}/voters/balances` endpoint did not paginate its results, so anyone could request the vote balances of every voter of a delegate in one API call. For delegates with large number of voters, this could overload the server before the rate limiter could catch it.

Solution: The API endpoint was removed.

Fixed in ARK Core? Yes on 2019-03-19 (`ArkEcosystem/core@e5d6a4a`)
Fixed in Solar Core? Yes on 2019-03-19 (`Solar-network/core@e5d6a4a`)

SECOND SIGNATURE TRANSACTIONS HALT THE NETWORK

A delegate node that received a second signature registration, immediately followed by a transaction signed with that second signature prior to the second signature registration being included in a block, would be unable to produce a new block as the second transaction would be accepted and added to its pool but would not validate in a block since the second signature registration had not yet been included in a block. As transactions were sent to all nodes, it would stop the blockchain as no delegate would be able to produce blocks.

Solution: Only accept a transaction with a second signature if the corresponding second signature registration has been included in a block.

Fixed in ARK Core? Yes on 2019-04-23 (`ArkEcosystem/core@9f298cd`)
Fixed in Solar Core? Yes on 2019-04-23 (`Solar-network/core@9f298cd`)

IMPROPER HANDLING OF BLOCKS WITH INVALID TRANSACTIONS

After receiving a block containing a transaction that could not be accepted, a node would correctly reject the block but would also incorrectly trigger a rollback. Since all nodes could receive this block simultaneously, triggering a network-wide rollback, it could permanently erase blocks and transactions.

Solution: Do not trigger a rollback operation after refusing a block.

Fixed in ARK Core? Yes on 2019-04-23 (`ArkEcosystem/core@9f298cd`)
Fixed in Solar Core? Yes on 2019-04-23 (`Solar-network/core@9f298cd`)

RACE CONDITION CAUSES INVALID BLOCKS

If a node received a high volume of transactions entering the pool in a short period, it would not filter out all the transactions from incoming blocks. This meant that when a delegate tried to produce a block, it may have included transactions already included in earlier blocks, which would then be rejected by the network, causing the affected delegates to repeatedly fail to produce blocks until their pools were manually cleaned or the transactions in the pool expired.

Solution: Check transactions in the pool and remove any that are no longer valid.

Fixed in ARK Core? Yes on 2019-05-28 (`ArkEcosystem/core@2cf81a0`)
Fixed in Solar Core? Yes on 2019-05-28 (`Solar-network/core@2cf81a0`)

SECOND SIGNATURE REGISTRATION MANIPULATION IN THE POOL

It was possible to stop the blockchain by manipulating second signature registrations. It involved broadcasting a transaction from a wallet without a second signature, then registering a second signature for that wallet which was included in a block prior to the initial transaction being included in a block.

Solution: Check that each transaction can still be applied before including them in a new block.

Fixed in ARK Core? Yes on 2019-05-30 (`ArkEcosystem/core@c8ca51f`)
Fixed in Solar Core? Yes on 2019-05-30 (`Solar-network/core@c8ca51f`)

BLOCK HEADER MANIPULATION IN QUORUM CALCULATIONS

Delegate nodes could be tricked into thinking they were producing blocks on multiple nodes in violation of the protocol, even when they were not. This

activated the automatic protection which stopped the nodes producing blocks. If all delegates were targeted, the blockchain would completely stop since no delegate would be able to produce blocks.

Solution: Keep producing blocks even if an over-height block is detected and check the heights and block headers of peers for validity.

Fixed in ARK Core? Yes on 2019-06-12 (`ArkEcosystem/core@72dc441`)
Fixed in Solar Core? Yes on 2019-06-12 (`Solar-network/core@72dc441`)

TRANSACTIONS IN BAD BLOCKS PURGE POOL TRANSACTIONS

When a block was rejected for failing verification, any transactions from the senders of any transactions in the block were purged from the pool, even if they failed verification. This meant anybody could create and broadcast a bad unverified block containing fake transactions to delete genuine transactions from that wallet from the pool.

Solution: Do not purge transactions from the pool when a block is not accepted.

Fixed in ARK Core? Yes on 2019-06-26 (`ArkEcosystem/core@d67fc61`)
Fixed in Solar Core? Yes on 2019-06-26 (`Solar-network/core@d67fc61`)

DELEGATES FORCED TO PRODUCE EMPTY BLOCKS

Invalid transactions could fill the pool to prevent real transactions being included in blocks as a node would retrieve pool transactions up to the maximum number of transactions per block, filter invalid ones and return the result. This could be empty after filtering if the pool returned more invalid transactions than the maximum number of transactions per block. This could also have evicted genuine transactions from the pool.

Solution: Keep trying to add valid transactions from the pool until the block is full or the pool has no more transactions.

Fixed in ARK Core? Yes on 2019-07-02 (`ArkEcosystem/core@0f19122`)
Fixed in Solar Core? Yes on 2019-07-02 (`Solar-network/core@0f19122`)

PEER-TO-PEER ENDPOINT SPAM

The `postTransactions` endpoint could be spammed with bundles of invalid transactions. The process of verifying all those transactions would overwhelm a node to prevent it being able to keep up with the network. It would stop receiving blocks, and delegates would be unable to produce blocks. Since the spam transactions were invalid, they would never be included in a block.

Solution: Add a 1 millisecond delay between processing each transaction to not tie up the event loop.

Fixed in ARK Core? Yes on 2019-07-30 (`ArkEcosystem/core@b12483b`)
Fixed in Solar Core? Yes on 2019-07-30 (`Solar-network/core@b12483b`)

PEER LISTS BECOMING TOO LARGE

There was no limit to the number of peers that could be added to peer lists. This could lead to extremely large peer lists being shared across peers which could not be processed in sufficient time, leading to delegates being unable to produce blocks as their nodes would be too busy processing the large number of peers. Furthermore, if IP addresses of non-Core peers were maliciously added in bulk, this could lead to all legitimate peers being used as an unintentional distributed denial of service network.

Solution: Limit the number of peers returned during peer discovery.

Fixed in ARK Core? Yes on 2019-07-30 (`ArkEcosystem/core@c34fd75`)
Fixed in Solar Core? Yes on 2019-07-30 (`Solar-network/core@c34fd75`)

LARGE PLUGIN NAMES

There was no limit to the length of plugin names returned in peer lists. This allowed a bandwidth flood attack by setting up a malicious node to send massive strings as plugin names resulting in overlength peer list responses that could not be processed in time due to their excessive size.

Solution: Introduce a stricter schema for peer replies.

Fixed in ARK Core? Yes on 2019-08-14 (`ArkEcosystem/core@29429be`)
Fixed in Solar Core? Yes on 2019-08-14 (`Solar-network/core@29429be`)

LACK OF SANITISATION FOR PEER-TO-PEER ENDPOINT REQUESTS

Spamming nodes with multiple simultaneous requests to invalid WebSocket endpoints on the peer-to-peer layer with large payload sizes created a memory leak which caused the socket workers to crash, stopping the node.

Solution: Disconnect peers when invalid requests are received.

Fixed in ARK Core? Yes on 2019-08-14 (`ArkEcosystem/core@29429be`)
Fixed in Solar Core? Yes on 2019-08-14 (`Solar-network/core@29429be`)

BLOCKING GENUINE REQUESTS BY EXCEEDING THE RATE LIMIT

Connections that exceeded the rate limit were gracefully disconnected by sending a `Forbidden` error message to the client. If a node was spammed with

requests exceeding the limit, the socket workers would continually send these messages back to the client, preventing it from handling genuine requests.

Solution: Terminate client connections immediately without a graceful close.

Fixed in ARK Core? Yes on 2019-09-04 (`ArkEcosystem/core@76f06ac`)
Fixed in Solar Core? Yes on 2019-09-04 (`Solar-network/core@76f06ac`)

MALFORMED PEER-TO-PEER MESSAGES CRASH NODES

It was possible to take down a node and either stop a delegate producing blocks or delay its block propagation to fork the network by exploiting a weakness in the peer-to-peer protocol by sending raw payloads to the node that did not conform to the required format or structure.

Solution: Disconnect peers when malformed messages are received.

Fixed in ARK Core? Yes on 2019-09-04 (`ArkEcosystem/core@5ac0c41`)
Fixed in Solar Core? Yes on 2019-09-04 (`Solar-network/core@5ac0c41`)

EXTERNALLY ACCESSING INTERNAL PEER-TO-PEER ENDPOINTS

Accessing an internal endpoint from an external unauthorised connection would send an `Unauthorised` error message to the client, but the socket would still be connected. As the connection remained open, in certain circumstances it was possible to trigger a node to continually send `Unauthorised` messages to block the socket workers, so they could no longer process legitimate requests.

Solution: Terminate client connections when not authorised.

Fixed in ARK Core? Yes on 2019-09-19 (ArkEcosystem/core@0dce189)
Fixed in Solar Core? Yes on 2019-09-19 (Solar-network/core@0dce189)

PING CONTROL FRAME BOMBARDMENT

When a WebSocket received a ping control frame, it would reply with a pong control frame. If a node was spammed with ping control frames, the socket workers would continually send pongs control frames back to the client, preventing it from handling genuine requests.

Solution: Terminate connections when receiving unwanted control frames.

Fixed in ARK Core? Yes on 2019-11-05 (ArkEcosystem/core@a8f88db)
Fixed in Solar Core? Yes on 2019-11-05 (Solar-network/core@a8f88db)

EMPTY HYBI WEBSOCKET HEADERS STOP NODES

It was possible to stop a delegate node producing blocks by continually sending payloads containing only a HyBi WebSocket header that signals a data frame, but one that contained no data. The effect was that the socket worker processes got stuck in a loop, consuming 100% CPU usage and could not complete their tasks fast enough to keep up with the network.

Solution: Terminate connections where the payload data size is below the minimum threshold for a valid message.

Fixed in ARK Core? Yes on 2019-12-09 (ArkEcosystem/core@94766ea)
Fixed in Solar Core? Yes on 2019-12-09 (Solar-network/core@94766ea)

MULTIPLE DISCONNECTION PACKETS CAUSE HIGH CPU USAGE

A user could maliciously craft a disconnection packet and constantly send it in a loop via the peer-to-peer network to overwhelm a node. As it was a valid packet, it did not get caught by any of the sanitisation checks, and the constant processing of these packets would cause the socket workers to consume all the CPU usage, effectively stalling nodes so they could not handle genuine traffic.

Solution: Terminate a connection after receiving a disconnection packet.

Fixed in ARK Core? Yes on 2019-12-19 (`ArkEcosystem/core@5c0c439`)
Fixed in Solar Core? Yes on 2019-12-19 (`Solar-network/core@5c0c439`)

BLOCKS CONTAINING TOO MANY TRANSACTIONS

A user could create and broadcast blocks containing thousands of transactions. As all the transactions inside the blocks were verified, this maxed out the CPU usage and prevented nodes operating correctly. It did not matter that the block ultimately failed verification or had a fake generator, as all transactions inside the block were still verified first. Since there were many thousands of transactions stuffed into the block, this CPU intensive process took too long to complete, and the rate limiter did not prevent the attack since the rate limit was reset long before the process finally finished, so an attacker could continue indefinitely to keep a node offline.

Solution: Discard and skip verification of blocks with too many transactions.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@4b43552`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@4b43552`)

PLAIN HTTP CONNECTIONS CRASH THE OPERATING SYSTEM

A node did not block any connections to any invalid HTTP paths accessed via the underlying HTTP server of the peer-to-peer layer, so a user could repeatedly send hundreds of thousands of plain HTTP connections to invalid paths and these connections would remain open, so a cluster of attack nodes could establish enough connections to use all available file descriptors on a server. This would crash the operating system completely since it would no longer be able to open any new files since all file descriptors were in use.

Solution: Terminate a connection if another is opened from the same IP address.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@a02d3ca`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@a02d3ca`)

TOO MANY CONNECTIONS CRASH THE OPERATING SYSTEM

An attacker could open many connections to a node because there was no filtering to prevent multiple connections per IP address. Each connection used a file descriptor in the operating system, and the number of available file descriptors is limited. An attacker could open enough simultaneous connections to use all the available open file descriptors on a node, which would crash it completely, since the operating system was no longer able to open any files.

Solution: Terminate a connection if another is opened from the same IP address.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@a02d3ca`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@a02d3ca`)

INVALID WEBSOCKET OPCODE HANDLING

Sending malformed WebSocket packets with reserved or unimplemented opcodes would trigger the `onerror` event handler, but ARK Core did not listen for this event and the connection was not blocked. The process of repeatedly throwing the error was sufficiently computationally expensive that it was possible to take down a delegate node and stop it producing blocks by sending a constant stream of malformed packets.

Solution: Terminate connections when invalid WebSocket opcodes are received.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@5b49532`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@5b49532`)

UNHANDLED EMITTED PEER-TO-PEER EVENTS

ARK Core incremented the rate limiter when the `emit` event was fired, but there were ways to create specially crafted payloads that would not trigger the event, so nodes could be flooded with such payloads to crash the socket workers.

Solution: Terminate connections accessing events without a defined handler.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@a324bf3`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@a324bf3`)

CPU INTENSIVE PAYLOADS WITH TOO MANY KEY-VALUE PAIRS

A user could continually send payloads containing too many key-value pairs to any peer-to-peer endpoint. This was too time consuming to parse, so a node would not be able to maintain synchronisation with the network, leading to delegates being unable to produce blocks, resulting in a stalled blockchain.

Solution: Terminate the connection if there are additional properties in the payload.

Fixed in ARK Core? Yes on 2020-01-16 (`ArkEcosystem/core@99fc278`)
Fixed in Solar Core? Yes on 2020-01-16 (`Solar-network/core@99fc278`)

LOCKING WALLET BALANCES TO PREVENT USERS TRANSACTING

The pool wallet balance did not always increase when receiving a transfer transaction with multiple recipients because `applyToRecipient` in `acceptChainedBlock` was only called if there was a matching `recipientId` already in the pool wallet manager. In the case of transfers with multiple recipients, the `recipientId` always matched the `senderId`, as the actual recipients were stored in a separate `asset` field instead, so this could be manipulated to lock the amount of the transfer to prevent the recipient accessing the funds because the pool wallet balance did not increase if the address of the recipient was present in the pool wallet manager and the address of the sender was not. This prevented the recipient from sending a transaction.

Solution: Always call `applyToRecipient` in `acceptChainedBlock`.

Fixed in ARK Core? Yes on 2020-03-04 (`ArkEcosystem/core@509db52`)
Fixed in Solar Core? Yes on 2020-03-04 (`Solar-network/core@509db52`)

LONG LIVED HTTP REQUESTS CRASH THE NODE

A peer could be bombarded with long-lived HTTP requests by opening a connection to the correct peer-to-peer path, but never upgrading the connection to a WebSocket in a Slowloris style attack. The connection could be kept alive by sending a HTTP header every second and never sending the `Upgrade: websocket`

or `Connection: upgrade` headers. The consequence was the ability to overload the node with open TCP sockets that were never established as WebSockets.

Solution: Terminate connections that are not quickly upgraded to WebSockets.

Fixed in ARK Core? Yes on 2020-03-12 (`ArkEcosystem/core@f1c35f3`)
Fixed in Solar Core? Yes on 2020-03-12 (`Solar-network/core@f1c35f3`)

OVERLOADING THE PUBLIC API

A user could stop a node processing transactions or blocks by opening simultaneous requests to the `/api/blocks` or `/api/transactions` endpoints.

Solution: Add additional indexes to the database tables and remove unnecessary sorting and filtering parameters for the Public API.

Fixed in ARK Core? Yes on 2020-03-19 (`ArkEcosystem/core@8929708`)
Fixed in Solar Core? Yes on 2020-03-19 (`Solar-network/core@8929708`)

SLOW NONCE COMPARISON WHEN GETTING POOL TRANSACTIONS

Filling the pool with transactions at a very fast rate meant that the call to `getUnconfirmedTransactions` would time out due to slow nonce comparison since too many transactions were in the pool, stopping block production for all delegates.

Solution: Use a sorted array for storing transactions by fee and nonce.

Fixed in ARK Core? Yes on 2020-04-28 (`ArkEcosystem/core@74aa1e0`)
Fixed in Solar Core? Yes on 2020-04-28 (`Solar-network/core@74aa1e0`)

POOL TREE STRUCTURE CAN EXCEED MAXIMUM CALL STACK SIZE

Filling the pool with transactions with an increasing fee of 1 satoshi caused the call to `getUnconfirmedTransactions` to fail as the maximum call stack size would be exceeded, causing the network to stall as no new blocks could be produced as the call to request the transactions to include in a block would never succeed.

Solution: Replace the tree memory structure with a sorted array.

Fixed in ARK Core? Yes on 2020-05-04 (`ArkEcosystem/core@e0b1431`)
Fixed in Solar Core? Yes on 2020-05-04 (`Solar-network/core@e0b1431`)

NO SANITISATION FOR MARSHALLED BLOCK CODEC PAYLOADS

Any payload using the block codec was not sanitised properly. The result was that additional objects could be added to the payload data if the new transport codec was used, which was exploitable to stop delegates producing blocks by overwhelming their servers with too many concurrent objects.

Solution: Add custom validation for the `postBlock` endpoint.

Fixed in ARK Core? Yes on 2020-05-12 (`ArkEcosystem/core@247b997`)
Fixed in Solar Core? Yes on 2020-05-12 (`Solar-network/core@247b997`)

INDUCED SLOW BLOCK PROPAGATION TO FORK THE NETWORK

Payloads could be constructed to target the Public API or the peer-to-peer layer which would cause a delegate to delay sending its newly produced block until the next block production slot. This could have caused a whole network fork, potentially stopping the entire blockchain, since it would have resulted in two conflicting blocks being propagated at the same height.

Solution: Discard blocks that were sent after the end of their slot.

Fixed in ARK Core? Yes on 2020-05-27 (`ArkEcosystem/core@36b5115`)
Fixed in Solar Core? Yes on 2020-05-27 (`Solar-network/core@36b5115`)

HIGH AND LOW S VALUES ACCEPTED IN ECDSA SIGNATURES

Transactions and blocks could have their signatures recalculated to generate a new ID while remaining cryptographically verified, since (*r, –s mod n*) is complementary to (*r, s*). This could be exploited to replay existing transactions and repeatedly cause network forks by recalculating the signatures of incoming blocks and broadcasting the recalculated blocks onwards, since this would lead to multiple different but valid blocks at the same height on the network.

Solution: Reject signatures where the *s* value is high.

Fixed in ARK Core? Yes on 2020-06-17 (`ArkEcosystem/core@54ba2d2`)
Fixed in Solar Core? Yes on 2020-06-17 (`Solar-network/core@54ba2d2`)

DELAYED COMPLETION OF PEER VERIFICATION

A user could stop the network by delaying the peer verification process when a delegate node was due to produce a block. As a delegate node enters its designated slot, it re-verifies its peers by sending a `getStatus` request to all of them. If a peer responds with a higher height than its own height, the node will query that peer further via `getBlocks` to fetch the blocks at the higher height to verify they are not forked. A malicious peer could respond with a deliberately higher height and then delay the ensuing response to the `getBlocks` request until the block production slot for the delegate was over. This would mean that the

peer verification would not complete, and the result was that the delegate would be unable to produce a block in time.

Solution: Cut loose from the peer verification process when the specified timeout duration is reached.

Fixed in ARK Core? Yes on 2020-06-17 (`ArkEcosystem/core@54ba2d2`)
Fixed in Solar Core? Yes on 2020-06-17 (`Solar-network/core@54ba2d2`)

BLOCK EXCEPTIONS VULNERABLE TO PREIMAGE ATTACKS

Block IDs can be added as exceptions which are accepted even if they fail verification, but there was no check to ensure that the correct transactions were inside the excepted blocks. This meant that transactions could be mutated or changed entirely inside of excepted blocks and the node would still accept them since it no longer mattered that the block would not pass verification checks. It could have been abused to poison the blockchain.

Solution: Check that the transactions inside the excepted blocks are correct.

Fixed in ARK Core? Yes on 2020-06-17 (`ArkEcosystem/core@54ba2d2`)
Fixed in Solar Core? Yes on 2020-06-17 (`Solar-network/core@54ba2d2`)

HALTING THE BLOCKCHAIN WITH BLOCK SCHEMA VIOLATIONS

It was possible to stop a node receiving new blocks by sending a single block which contained a chained block header with a `numberOfTransactions` value greater than 0 but where no transactions were included in the block. This caused an error to be thrown which stopped the processing queue indefinitely since the processing queue was not able to recover. If such a block was sent to the whole network, the entire blockchain would halt.

Solution: Ensure the processing queue continues when errors are thrown.

Fixed in ARK Core? Yes on 2020-06-17 (`ArkEcosystem/core@54ba2d2`)
Fixed in Solar Core? Yes on 2020-06-17 (`Solar-network/core@54ba2d2`)

PREPENDING ZEROS IN ECDSA SIGNATURE COMPONENTS

In cryptographic verification, the *r* and *s* components of ECDSA signatures are integers, but the hashing process used by ARK Core to calculate the ID of a transaction or block uses their byte sequence instead. ARK Core did not check for the presence of extra zeros at the start of either the *r* or *s* components of a signature, so prepending extra zeros to either component would change the ID of any block or transaction since this would modify the byte sequence, while remaining cryptographically verified as the extra zeros were ignored during verification since that process uses the integer values instead. This meant transactions could be replayed since they would have had new IDs, and the network was vulnerable to repeated unlimited forking if multiple blocks were propagated at the same height with different block IDs.

Solution: Reject signatures with prepended zeros in either component.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@1b0863c`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@1b0863c`)

POOL POISONING WITH INVALID TRANSACTIONS

There was a low limit on the number of transactions returned from the pool for inclusion in a new block prior to filtering them. A bad actor could repeatedly spam the pool with hundreds or thousands of invalid transactions with a very high fee - which, as they were invalid, would never be included in blocks, so the

actual cost was zero - but the high fee meant they would take precedence over all genuine transactions in the pool, so after filtering, the block would be empty.

Solution: Fetch more transactions from the pool to account for the possibility of invalid transactions being filtered.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@0450ee8`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@0450ee8`)

BANDWIDTH FLOOD DUE TO HTTP GET REDIRECTION

A malicious HTTP `GET` request could be sent to the peer communicator to redirect traffic to different server during port ping operations. This could have, among other possibilities, downloaded very large multi-gigabyte files to consume all the bandwidth of a node and potentially violate service provider policies or trigger overage charges.

Solution: Use `HEAD` requests without redirects instead of `GET` for port pings.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@db226bd`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@db226bd`)

VERY SLOW LARGE CUMULATIVE BLOCK DOWNLOAD PAYLOADS

Filling enough blocks with transactions to make a large cumulative payload and then repeatedly calling the `getBlocks` peer-to-peer endpoint to download that set of blocks from multiple IP addresses at the same time every second would prevent delegates producing blocks due to the amount of processing time needed to construct the payload, causing significantly elevated CPU usage. This could be amplified as new requests could be made before the previous requests finished, stacking more and more requests until the database was overwhelmed.

Solution: Add a stricter rate limit for the `getBlocks` endpoint.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@df7a3aa`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@df7a3aa`)

CONSECUTIVE BIG BLOCKS EXCEED MAXIMUM PAYLOAD LIMIT

By repeatedly sending large transactions to fill consecutive blocks, the maximum client-side payload limit could be exceeded for any WebSocket connection. This would trigger peer bans when trying to download a batch of blocks that exceeded this payload size and meant no new nodes could ever join the network as they would not be able to download a batch of blocks exceeding this limit, thus preventing any new relays syncing with the network.

Solution: Reduce the number of requested blocks when peers return no blocks.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@e223287`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@e223287`)

DER SIGNATURE MANIPULATION

Blocks and transactions signed with DER-encoded signatures could be manipulated by appending extra data to the end of the signature outside of the *r* and *s* values. This meant the transactions and blocks were still cryptographically valid but would have a different ID, allowing for transaction replay and to persistently fork the network.

Solution: Reject signatures with extra data outside of the *r* and *s* values.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@1b0863c`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@1b0863c`)

NEGATIVE VALUES ACCEPTED IN ECDSA SIGNATURES

ARK Core did not check for negative values in the *r* or *s* components of ECDSA signatures, which are not allowed in the specification. It was possible to maliciously modify an existing valid signature to include negative values for either *r* or *s* and this signature would still erroneously verify as `true`, since the values were internally normalised as positive integers. However, by doing so, the block or transaction would have a different ID, potentially leading to transaction replay or network forks.

Solution: Reject signatures containing negative *r* or *s* values.

Fixed in ARK Core? Yes on 2020-07-22 (`ArkEcosystem/core@0783ec0`)
Fixed in Solar Core? Yes on 2020-07-22 (`Solar-network/core@0783ec0`)

OUT OF MEMORY CRASHES DUE TO HTTP HEADER MANIPULATION

ARK Core could be coerced to download any arbitrary file from a remote server by transmitting a `303 See Other` header response to the peer communicator. If the HTTP header signalled that the payload was compressed, it was automatically decompressed and stored in memory, potentially expanding to many times its original size. A specially crafted compressed file could grow too large, triggering an out of memory error which would crash the node.

Solution: Replace `GET` requests with `HEAD` requests so payloads are not downloaded, since only the response headers are required.

Fixed in ARK Core? Yes on 2020-09-09 (`ArkEcosystem/core@da13465`)
Fixed in Solar Core? Yes on 2020-09-09 (`Solar-network/core@da13465`)

NO MAXIMUM PAYLOAD LIMIT FOR NEW PEER CONNECTIONS

ARK Core dynamically adjusts the maximum payload size of a peer response depending on the type of request being made to the peer, however it did not set a default maximum payload size when initially establishing a connection to a peer. This allowed an attacker to send a very large payload as soon as the connection was opened prior to any request being made, which would crash a node by causing an out of memory condition.

Solution: Set a sane maximum payload limit for all new peer connections.

Fixed in ARK Core? Yes on 2020-09-17 (`ArkEcosystem/core@3ac3eb1`)
Fixed in Solar Core? Yes on 2020-09-17 (`Solar-network/core@3ac3eb1`)

INSUFFICIENT TRANSACTION ASSET VALIDATION

There was a flaw in the schema validation of incoming multi-recipient transfer transactions which meant an attacker could add extra assets to a transaction while still passing validation checks. Adding too many of these additional assets would cause delays in parsing the transaction, leading to backlogged requests, timeouts and in the case of delegate nodes, an inability to produce blocks.

Solution: Stricter checking of incoming multi-recipient transfer transactions.

Fixed in ARK Core? Yes on 2020-09-17 (`ArkEcosystem/core@620027d`)
Fixed in Solar Core? Yes on 2020-09-17 (`Solar-network/core@620027d`)

PEER LISTS CAN EXCEED THE MAXIMUM ALLOWED PAYLOAD SIZE

Peer lists sent via the peer-to-peer protocol contained data about the plugins each peer was running. This meant a malicious user was able to construct a

swarm of peers all configured with multiple plugins, so the cumulative size of peer lists would grow to exceed the maximum permitted size of 100KiB. This would prevent any node receiving an updated peer list, stopping new nodes joining the network and would disconnect existing ones when receiving an oversized peer list.

Solution: Only send IP addresses in peer lists over the peer-to-peer protocol.

Fixed in ARK Core? Yes on 2020-09-23 (`ArkEcosystem/core@6988cdd`)
Fixed in Solar Core? Yes on 2020-09-23 (`Solar-network/core@6988cdd`)

INADEQUATE RATE LIMITING ON OUTGOING SOCKETS

Although incoming sockets were properly rate limited, outgoing sockets were not, as the only rate limit applied to outgoing sockets was for internal ping messages. Once a node made an outgoing connection to a malicious peer, that peer could continually send packets of data to overwhelm the node if the data was anything other than an internal ping message.

Solution: Add rate limiting to outgoing sockets.

Fixed in ARK Core? Yes on 2020-09-23 (`ArkEcosystem/core@6988cdd`)
Fixed in Solar Core? Yes on 2020-09-23 (`Solar-network/core@6988cdd`)

OUTGOING CONNECTIONS NOT ALWAYS DESTROYED PROPERLY

An outgoing socket connection was directly terminated after receiving an unsupported WebSocket frame without notifying the underlying client framework that the connection was deliberately closed. This meant the framework would automatically attempt to reconnect to the peer again, so a

malicious user could continue sending more unsupported WebSocket frames in a denial of service attack.

Solution: Destroy connections properly after terminating them.

Fixed in ARK Core? Yes on 2020-09-28 (`ArkEcosystem/core@a853d63`)
Fixed in Solar Core? Yes on 2020-09-28 (`Solar-network/core@a853d63`)

LOCKING POSTGRESQL BY REQUESTING VERY HIGH HEIGHTS

PostgreSQL could be jammed at 100% CPU usage across multiple cores by requesting one or more blocks from a node starting at an exceptionally high height. This request took longer to complete than the rate limit for the relevant endpoint, so a bad actor could keep sending requests constantly to lock up PostgreSQL at 100% CPU usage across multiple cores, preventing normal node operations and stopping delegates producing blocks.

Solution: Return an empty array without hitting the database if the requested height is greater than the current height.

Fixed in ARK Core? Yes on 2020-10-22 (`ArkEcosystem/core@312899c`)
Fixed in Solar Core? Yes on 2020-10-22 (`Solar-network/core@312899c`)

BINARY DATA PAYLOADS STOP BLOCK PRODUCTION

WebSocket transmissions can be either text or binary, and ARK Core historically only used textual strings, but there was no filter to block binary payloads in the peer-to-peer layer. It was therefore possible to send binary payloads which would incur large overheads when automatically unmarshalling the binary buffer which would tie up the socket workers to prevent a delegate node obtaining the correct network state, rendering it unable to produce blocks.

Solution: Ensure incoming WebSocket messages are the correct type.

Fixed in ARK Core? Yes on 2020-10-22 (`ArkEcosystem/core@5e1c5a3`)
Fixed in Solar Core? Yes on 2020-10-22 (`Solar-network/core@5e1c5a3`)

BIG PAYLOADS ON INTERNAL ENDPOINTS STOP NODE OPERATION

A malicious user could take advantage of the lack of schema validation on internal endpoints by sending a large payload to any of these endpoints which is slow to parse. Although the connection would be terminated because the user was not authorised to access the endpoint, the IP address would not be banned, so a bad actor could keep reconnecting and sending this payload in a loop which would jam the socket workers.

Solution: Ban unauthorised connections trying to access any internal endpoint.

Fixed in ARK Core? Yes on 2020-10-22 (`ArkEcosystem/core@dd96ee4`)
Fixed in Solar Core? Yes on 2020-10-22 (`Solar-network/core@dd96ee4`)

BLOCKS ACCEPTED BUT NOT PROPAGATED OUT OF SLOT

A block was only propagated onwards to other peers if the current slot time was less than or equal to the block timestamp. This meant it was possible to delay the broadcast of a block until the next slot began so receiving nodes would accept it but not broadcast it further. This could cause significant problems for the network with quorum calculations and over-height block headers.

Solution: Always broadcast the last block.

Fixed in ARK Core? Yes on 2020-11-17 (`ArkEcosystem/core@c291166`)
Fixed in Solar Core? Yes on 2020-11-17 (`Solar-network/core@c291166`)

EVADING RATE LIMITER BANNING

If a request exceeded the rate limit on a specific endpoint, the connection was closed but not blocked. By continually targeting endpoints with restrictive rate limits in this manner, they would never hit the global rate limit which triggered a ban, so they were never blocked. Consequently, they could continue connecting and sending more requests to tie up the socket workers to ultimately stop a node functioning correctly and stop delegates producing blocks.

Solution: Block a connection when the rate limit is exceeded on any endpoint.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@eecc40c`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@eecc40c`)

SLOW QUERY REQUESTING BLOCKS FROM SPECIFIC DELEGATES

The `/api/delegates/{delegate}/blocks` endpoint could trigger a slow query when requesting a list of blocks. Those requests could be parallelised to overwhelm the database server and make the node unresponsive.

Solution: Add a new `(generatorPublicKey, height)` index to the `blocks` table in the PostgreSQL database.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@6fb4ab9`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@6fb4ab9`)

DENIAL OF SERVICE VIA REVIVER FUNCTION IN TRANSPORT CODEC

Every message that was sent or received over the peer-to-peer protocol was processed by a custom codec. Part of the decoding process in that codec attempted to parse the received message string with a reviver function. If that

function failed for any reason, the rate limiter was not triggered, so a malicious user could execute a denial of service attack by continually and uninterruptedly sending payloads that caused the reviver function to fail.

Solution: Parse a message without the reviver function if the reviver failed so the rate limiter will still catch it.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@8e41f06`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@8e41f06`)

INCOMING CONNECTIONS NOT BANNED IF BASIC CHECKS FAIL

Requests to the socket server needed to adhere to a basic structure containing `data` and `headers` properties, which must be objects nested inside a root `data` object. If the request did not comply with this specification, the connection was terminated but the IP address was not banned. This meant a malicious user could continuously reconnect and keep sending non-conformant requests in a tight loop, using up CPU time.

Solution: Ban connections not conforming to the required basic structure.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@eecc40c`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@eecc40c`)

SOCKET EVENT LISTENERS NOT REATTACHED ON RECONNECTION

When a new connection is made by the peer connector, a node would attach event listeners to the underlying socket to handle errors, abnormal packets and rate limiting. However, these listeners were only attached to the initial socket created for the connection. If the connection was lost and later reconnected, a new socket was created without reattaching these event listeners.

Solution: Attach the event listeners inside the `connecting` event of the socket.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@b562592`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@b562592`)

UNPUNISHED SCHEMA VIOLATION REQUESTING COMMON BLOCKS

The `getCommonBlocks` endpoint schema requires an object containing an array of block ID strings. If the array contained items other than strings, an error was returned to the client, but the connection remained open. This allowed a user to keep sending a malicious payload to this endpoint without being banned.

Solution: Ban connections that violate the schema.

Fixed in ARK Core? Yes on 2020-11-18 (`ArkEcosystem/core@2b39363`)
Fixed in Solar Core? Yes on 2020-11-18 (`Solar-network/core@2b39363`)

INSUFFICIENT VALIDATION OF MULTI-RECIPIENT TRANSFERS

There were no checks to ensure the `amount` and `recipientId` values for incoming multi-recipient transfer transactions were the correct data types, so an attacker could set their values to large objects which would take a long time to parse. This processing delay would prevent a node staying synchronised with the network.

Solution: Check the data types are correct before parsing the transaction.

Fixed in ARK Core? Yes on 2020-12-03 (`ArkEcosystem/core@29eaf10`)
Fixed in Solar Core? Yes on 2020-12-03 (`Solar-network/core@29eaf10`)

UNSANITISED CLIENT-SIDE DISCONNECTION PAYLOADS

No sanitisation was performed when a client-side graceful disconnection payload was received, so an attacker could add many extra irrelevant properties to this payload which were passed through unchecked to the transport codec and framework for more parsing and processing. This was time consuming and could cause the socket workers to use up all CPU cycles, rendering a node unable to function, and in the event of an active delegate, it would miss blocks.

Solution: Ban connections if any extra properties are present.

Fixed in ARK Core? Yes on 2020-12-03 (`ArkEcosystem/core@c6d7421`)
Fixed in Solar Core? Yes on 2020-12-03 (`Solar-network/core@c6d7421`)

CALL ID IN PEER-TO-PEER REQUESTS SET TO ANY VALUE

Every peer-to-peer request requiring a response needed to include a call ID value which was also included in the response so the client would know which request the response was for. Although sequentially incrementing integers were used for the call ID, non-integer values were not prohibited. A malicious user could send a request with a large object as the call ID, which would stop a node operating correctly as its socket workers would be busy processing the large object.

Solution: Enforce a schema check to ensure the call ID value is an integer.

Fixed in ARK Core? Yes on 2020-12-03 (`ArkEcosystem/core@4565eba`)
Fixed in Solar Core? Yes on 2020-12-03 (`Solar-network/core@4565eba`)

DUPLICATE KEYS IN PEER-TO-PEER REQUESTS TAKE DOWN A NODE

An attacker could add a large object as a property to any peer-to-peer request, then add a second property with the same name but with a valid value. This still passed validation checks since the latter value replaced the initial value, but the extra processing incurred by the initial value caused the socket workers to become stuck at maximum CPU capacity, rendering a node unable to function.

Solution: Ban connections with multiple identical property names.

Fixed in ARK Core? Yes on 2020-12-03 (`ArkEcosystem/core@ae8dcd7`)
Fixed in Solar Core? Yes on 2020-12-03 (`Solar-network/core@ae8dcd7`)

LONG FORM LENGTH VALUES ACCEPTED IN ECDSA SIGNATURES

DER-encoded ECDSA signatures allow signature lengths to be expressed in long or short form. Swapping an existing short form length to the long form would mutate the ID without invalidating the signature, leading to transaction replay or network forking if two valid blocks - one with the short form length and the other with the long form length - were propagated at the same height.

Solution: Reject signatures containing long form length values.

Fixed in ARK Core? Yes on 2020-12-03 (`ArkEcosystem/core@8f8976d`)
Fixed in Solar Core? Yes on 2020-12-03 (`Solar-network/core@8f8976d`)

RATE LIMIT EVASION VIA HTTP HEADER MANIPULATION

The rate limiter used the `x-forwarded-for` header, if it existed, to determine the IP address of each connection to enforce rate limits, rather than the IP address of the connecting socket. This permitted a malicious user to either bypass the

rate limiter or deny service to legitimate IP addresses by deliberately exceeding the rate limit using a spoofed IP address.

Solution: Use `request.info.remoteAddress` from the Hapi server to get the IP address of the requester rather than relying on the `X-Forwarded-For` header, unless the `trustProxy` directive is set to `true`.

Fixed in ARK Core? Yes on 2020-12-11 (`ArkEcosystem/core@fff3ecb`)
Fixed in Solar Core? Yes on 2020-12-11 (`Solar-network/core@fff3ecb`)

PEER LISTS RETURN MORE PEERS THAN ALLOWED BY SCHEMA

Peers always returned their entire peer lists when requested via `getPeers`, but the reply schema enforced a maximum of 2,000 peers in the response. It was possible for a malicious user to set up enough peers to exceed this amount, so node peer lists would contain more than 2,000 entries. This meant that new peers would not be able to join the network as they would reject the incoming peer lists, and existing nodes would be unable to receive updated peer lists, eventually disconnecting when receiving an oversized peer list.

Solution: Cap peer list responses to the maximum allowed number of peers.

Fixed in ARK Core? Yes on 2020-12-11 (`ArkEcosystem/core@4c79c45`)
Fixed in Solar Core? Yes on 2020-12-11 (`Solar-network/core@4c79c45`)

TOO HIGH TIMEOUT VALUE WHEN DOWNLOADING BLOCKS

The timeout value for downloading blocks was 10 minutes, meaning a node would wait 10 minutes for a response before trying another peer. This meant an attacker could halt a node for 10 minutes on boot or during synchronisation by deliberately not sending a response to a `getBlocks` request.

Solution: Use a saner timeout value of 30 seconds.

Fixed in ARK Core? Yes on 2020-12-11 (`ArkEcosystem/core@9370175`)
Fixed in Solar Core? Yes on 2020-12-11 (`Solar-network/core@9370175`)

MAXIMUM PAYLOAD SIZE NOT RESET IN PEER CONNECTOR

When a node issued a request to a peer, it would set the maximum payload size according to the type of request being made but it did not reset the value once the response had been received. A malicious peer could bombard the socket with large payloads once a higher maximum payload size had been set.

Solution: Reset the maximum payload size after every response.

Fixed in ARK Core? Yes on 2020-12-11 (`ArkEcosystem/core@f1bf195`)
Fixed in Solar Core? Yes on 2020-12-11 (`Solar-network/core@f1bf195`)

ADDITIONAL PROPERTIES ALLOWED IN TRANSACTIONS

The `postTransactions` endpoint permitted additional objects inside the root `data` object in addition to the `transactions` array. This allowed a user to send a complex payload which consumed too many resources to parse, potentially resulting in missed blocks and loss of network synchronisation.

Solution: Do not permit additional objects inside the payload.

Fixed in ARK Core? Yes on 2020-12-11 (`ArkEcosystem/core@021d18a`)
Fixed in Solar Core? Yes on 2020-12-11 (`Solar-network/core@021d18a`)

PERMESSAGE-DEFLATE ENABLED IN PEER CLIENT CONNECTIONS

The `permessage-deflate` WebSocket compression option is known to cause catastrophic memory fragmentation and there is such a disclaimer on the `ws` module used by the underlying framework for peer-to-peer communication. Although server-side `permessage-deflate` was disabled, it was enabled client-side, so a malicious user could set up servers with `permessage-deflate` enabled and the outbound connections would use `permessage-deflate` compression. A malicious user could then request large, compressed payloads to build up memory fragmentation until the node stopped responding. Delegate nodes would then be unable to produce blocks, potentially halting the network.

Solution: Explicitly disable `permessage-deflate` in peer client connections.

Fixed in ARK Core? Yes on 2021-07-08 (`ArkEcosystem/core@7a6d198`)
Fixed in Solar Core? Yes on 2021-07-08 (`Solar-network/core@7a6d198`)

DENIAL OF SERVICE VIA TRANSACTION FLOODING

Spamming the `postTransactions` endpoint with transactions, waiting 2 seconds and repeating this in a loop would cause CPU usage to spike to 100%+ so a node could no longer process blocks fast enough to maintain synchronisation with the network. Eventually, the rate limiter would disconnect the attacker, but they could reconnect up to 4 times in a 30 second window even with the `iptables` script active which was sufficient to take down a node.

Solution: Use a more efficient custom byte buffer to deserialise transactions.

Fixed in ARK Core? Yes on 2022-01-06 (`ArkEcosystem/core@22dffb4`)
Fixed in Solar Core? Yes on 2022-01-06 (`Solar-network/core@22dffb4`)

INCORRECT ENDIANNESS OF TRANSACTION BUFFER

The `typeGroup` of any transaction sent over the `postTransactions` endpoint was incorrectly deserialised due to an issue with endianness. The buffer read the little-endian data as big-endian, so the `typeGroup` was decoded wrongly. This meant the `workerPool.isTypeGroupSupported` check always returned `false`, so the transaction skipped the worker pool. Consequently, all requests hit the main thread, so spamming the endpoint would stop it processing blocks. Although the rate limiter would eventually disconnect the attacker, they could reconnect up to 4 times in a 30 second window even with the `iptables` script active which was sufficient to take down a node.

Solution: Ensure the buffer uses the correct endianness.

Fixed in ARK Core? Yes on 2022-01-06 (`ArkEcosystem/core@22dffb4`)
Fixed in Solar Core? Yes on 2022-01-06 (`Solar-network/core@22dffb4`)

HTTP CONNECTIONS TO THE PEER-TO-PEER SERVER KEPT ALIVE

The HTTP server used for peer-to-peer connections did not drop a connection after receiving an invalid HTTP request. It responded with a `4xx` error code but kept the connection alive, so users could continue sending requests in a loop.

Solution: Close the connection when an invalid request is received.

Fixed in ARK Core? Yes on 2022-03-24 (`ArkEcosystem/core@3e4ffed`)
Fixed in Solar Core? Yes on 2022-03-23 (`Solar-network/core@568b545`)

WALLET REPOSITORY POLLUTION VIA PUBLIC API ENDPOINTS

API endpoints such as `/api/wallets/<address>/transactions` or `/api/transactions?recipientId=<address>` were vulnerable to wallet repository pollution where `address` was a non-existent wallet address. These endpoints triggered calls to `findByAddress` or `findByPublicKey` which would create a new empty wallet if an existing wallet did not exist, which was then perpetually stored in the wallet repository, taking up memory. With enough time and resources, enough empty wallets could be added to use up all memory of the node since this was a free attack vector as no transactions were required.

Solution: Check `hasByAddress` or `hasByPublicKey` for the address and abort if it returns `false`, that way `findByAddress` or `findByPublicKey` are only called if the address already exists in the wallet repository.

Fixed in ARK Core? Yes on 2022-08-03 (`ArkEcosystem/core@38c4c91`)
Fixed in Solar Core? Yes on 2022-07-28 (`Solar-network/core@23ac786`)

WALLET REPOSITORY POLLUTION VIA PEER-TO-PEER REQUESTS

The `postTransactions` peer-to-peer endpoint could be used to send an invalid transfer transaction to a new empty wallet. This would create a new entry in the wallet repository for the recipient. Since the transaction would fail, the wallet would remain empty, but it was not removed from the wallet repository. Repeating this would pollute the wallet repository with enough empty wallets to cause memory exhaustion. The wallet repository also did not purge wallets that became empty after reverting blocks that contained the only existing transactions on the blockchain that referenced them.

Solution: Re-index wallets involved in a transaction after applying or reverting it.

Fixed in ARK Core? Partial on 2022-08-08 (`ArkEcosystem/core@5df7060`)
Fixed in Solar Core? Yes on 2022-08-03 (`Solar-network/core@6b21e89`)

POOL WALLET NONCE CORRUPTION LOCKING USER BALANCES

The pool did not clear invalid transactions, leading to incorrect nonces in pool wallets which prevented affected wallets from sending any new transactions, de facto blocking them from accessing their coins. This occurred when a transaction was submitted that was initially valid, ergo it entered the pool and incremented the in-pool nonce value for the sender wallet, but prior to its inclusion in a block, a different transaction was instead included in a block that invalidated the original transaction, meaning the original transaction could never be included in a block, yet the invalid transaction was retained in the pool.

Solution: Revalidate transactions in the pool after every block, removing any that are no longer valid.

Fixed in ARK Core? Yes on 2022-09-30 (`ArkEcosystem/core@a96128c`)
Fixed in Solar Core? Yes on 2022-09-08 (`Solar-network/core@c710754`)