Open Source Technology Improvement Fund

Building Stronger Open Source Apps for the World

Four Audits of RandomX for Monero and Arweave have been Completed – Results

August 10, 2019 ostifadmin

As always, remember that our work only happens with the support of our sponsors and the community. Consider donating to the cause and getting the companies that you work at and patronize to get involved. We are always short on funding and more money always means more research. Special thank you's to Private Internet Access, Monero Research Lab, NordVPN, ExpressVPN, and Mullvad for contributing to OSTIF and helping us move the open-source world forward.

Categories

Audits
Bug Bounties
Encryption
Financial
Fundraiser
Kudelski Security
Monero
News
Open Source
OpenSSL
OpenVPN
QuarksLab
Security
Trail of Bits
Transparency
Unbound DNS
Uncategorized
VeraCrypt
WireGuard

with funds from the Monero community and Monero Research Lab.

We have been working with the Monero community, Arweave, Monero Research Lab, and four audit teams to conduct a deep an thorough review of RandomX, a new proof of work algorithm that aims to push the “egalitarian ideal” for cryptocurrency.

RandomX aims to make highly-optimized custom hardware slower or far less efficient than general purpose computer hardware that is available to everyone. The thinking behind this move is that hardware that is available to everyone is far more likely to be used by the general populace and enthusiasts, which hedges against centralization of the computing power that is verifying cryptocurrency transactions.

One of the classic problems with cryptocurrency is that if any one party (or a collaborating cartel of malicious parties) were to possess at least 51% of the computing power of the network, they can re-write transactions to duplicate currency, invalidate real transactions, or otherwise manipulate the network. The rise of custom equipment (ASICs and FPGAs) has given groups that are willing to commit those resources majority stakes in this computing power (aka hashing power).

RandomX aims to solve this problem by making the algorithm too complex for ASICs and FPGAs to effectively optimize. They will either be significantly slower, too costly to build due to complexity, or too expensive to run compared to general purpose hardware (in electricity costs).

RandomX is a brand-new idea, but it employs a lot of well known cryptographic functions and applies them in interesting ways.

In order to verify the security properties of RandomX, Monero Research Lab, The Monero Community, and Arweave have all contributed significantly to sponsor security research that we all coordinated together.

These four audits were done in a series, with the results and fixes from each audit feeding into the next one. This series of cascading audits gave us perspectives from different teams as well as opportunities to look at aspects of code that others didn’t focus on, leading to a thorough overall security review.

Archives

December 2019
August 2019
July 2019
June 2019
May 2019
February 2019
January 2019
October 2018
September 2018
July 2018
May 2018
March 2018
January 2018
November 2017
October 2017
September 2017
July 2017
June 2017
May 2017
April 2017
March 2017
February 2017
January 2017
December 2016
November 2016
October 2016
September 2016
August 2016
June 2016
May 2016

The goal of these audits is to determine not only that the coding and cryptography are error free, but that the implementation does not easily allow an ASIC or FPGA design to function with a significant advantage over general-purpose hardware.

The Broad Strokes

All four security teams agree that the RandomX code is of very high quality, but that the documentation could be improved in order to prevent implementation mistakes in non-Monero and non-Arweave blockchains. None of the four teams found cryptographic short cuts in the code that would allow a significant speedup in hashing power due to a bug, nor functions or components of RandomX that could be easily bypassed with hardware optimization (ASIC or FPGA). The consensus is that the code is near deployment-ready, and that over time some small coding practice changes and documentation improvements can further improve safety as RandomX evolves and changes over time with updates. Some suggestions to further harden RandomXs resistance to hardware optimization have also been detailed.

FIXED = A patch was applied that corrects this issue.
SOLVED = Clarification given by the RandomX team indicates that the issue does not need patching for other reasons.
DISPUTED = The RandomX team disagrees with the assessment.

If an issue does not have a status, it is because the issue is still being discussed between the teams. It does not indicate a dispute, only that OSTIF does not know the current status. The status of the issues will be updated here as we learn more about them.

Audit 1 – Trail of Bits sponsored by Arweave

Issue 1 – FIXED – Single Round Used in AesGenerator
RandomX was using a single round of AES in order to achieve diffusion. You need at least two rounds of AES to fully mix the source data. This is problematic because in an exploit scenario, if an attacker were able to find a similar bias in other components of RandomX, pieces of the algorithm could be effectively skipped

February 2016
January 2016
December 2015
November 2015
October 2015
September 2015
July 2015
May 2015

present across many RandomX functions, which was not the case. Nevertheless, RandomX has now been updated to use AesGenerator4R, which performs four rounds of AES to properly mix and diffuse the data.

Issue 2 – FIXED – Insufficient Testing and Validation of VM Correctness
RandomX uses a Virtual Machine, but does not have a machine-readable specification to test against. This is important because as the implementation changes, very small changes can lead to forks at unpredictable locations in the blockchain. Being able to test against a specification detects these (potentially rare and hard to detect) problems before they become an issue on the live-chain. Testing has been significantly expanded, with unit testing available for every RandomX insruction.

Issue 3 – FIXED – RandomX Configurable Parameters are Brittle – Pull Request 59
RandomX has a lot of configuration options, in order to allow multiple cryptocurrency projects to implement RandomX. Some of these options can significantly damage the security and ASIC resistance properties of RandomX. It is strongly recommended that at the very least comments are added to parameters that have a significant impact on the security and robustness of RandomX, and it should be a long term goal to remove the ability to choose unsafe parameters entirely.The RandomX team has added checks to make sure that value are sane, and they have also added significantly more documentation about which values to select.

Additional Miscellany
Some code quality improvement suggestions. Suggested improvements to randomness in SuperscalarHash. Trail of Bits also did a complex randomness analysis of RandomX and found it select various paths sufficiently, which prevents optimizing for specific paths that are selected more often via a bias. They also provided an excellent resource on which tunable parameters are safe to change in RandomX and which should be left alone. (Section E.) They also recommended specific values for Arweave, the sponsor of Trail of Bits' research.

The full Arweave paper can be found here.

sponsored by OSTIF, Monero Research Lab, and the Monero Community

(Redundant findings from Audit 1 by Trail of Bits have been omitted.)

Issue 1 – SOLVED – Usage of Unoptimized BLAKE2
RandomX's version of the BLAKE2 hash function is unoptimized and does not take advantage of AVX512 CPU optimization. A speedup of up to 40% can be achieved by using an optimized version of BLAKE2. However, the overall performance of RandomX is not dependent on BLAKE2, so the only real advantage of improving this code is power consumption / cpu time. The RandomX team is exploring the use of LibSodium's optimized implementation of BLAKE2 for a future release.

Issue 2 – FIXED – Lack of NULL Checks – Pull Request 88
RandomX does not perform NULL checks when dereferencing the pointer in most areas of the code. NULL pointer dereferences are a common security problem in languages that allow them. It is generally a good practice to maintain null checks even if you are confident in your code, because errors can be introduced later through updates that null checks will catch.

Issue 3 – SOLVED – randomx_reciprocal() Could Divide by Zero – Pull Request 88
Dividing by zero can cause crashes and other undesired behavior. After discussion with the development team, other areas of code make sure that this value cannot be zero. A check was added anyway in pull request 88.

Issue 4 – FIXED – randomx.cpp Has a Type Mismatch Error – Pull Request 88
This type of error can lead to an attacker using crafted inputs to force undesired behavior (an integer overflow). The RandomX team added checks and asserts to the code in order to cause the application to crash if these invalid inputs are encountered.

Issue 5 – FIXED – getCodeSize() Has a Potential Integer Underflow – Pull Request 95

Hash errors could cause RandomX to enter an undesired state, allowing a user to bypass some steps in RandomX. Upon further investigation, there is a check for this at compile-time, but an extra layer of protection has been added to RandomX by the development team to manually check this value and to throw an assert (and crash the application) if an error is encountered.

Issue 7 – SOLVED – Potential Out-of-Bound Writes in fillAes1Rx4() and fillAes4Rx4() – Pull Request 95
This was a non-issue as there are bounds checks at compile-time. Additional checks were added for safety.

Issue 8 – SOLVED – isPowerOf2() Can Have an Input of Zero, Which is not a Power of 2 – Pull Request 95
A value check was mislabeled as "isPowerOf2()" and a value of Zero could be called into the function, causing potentially undesired behavior. After talking with the RandomX team, the ability to call a zero into the function was intended, but the function was renamed for better clarity on what the check is actually doing. It is now called isZeroOrPowerOf2().

Audit 3 – X41 D-Sec sponsored by OSTIF, Monero Research Lab, and the Monero Community

Issue 1 – FIXED – Hard Coded Code Size in JIT Compiler – Pull Request 98
The RandomX JIT Compiler is hard coded to a size of 64k, which can be exceeded by changing some RandomX settings (this would affect non-Monero implementations). This could have both security and stability implications.

Issue 2 – SOLVED – Integer Handling in Jump-Target Calculation
The jump-target function does not consider edge cases that could lead to Integer Overflows or Integer Wraparounds. After talking with the RandomX team, this appears to be a non-issue because there is a small range of valid inputs that prevent this issue from arising.

DatasetSize can overflow to a very small number in 32-bit systems, due to the size limit of memory allocation for 32-bit addressing. After speaking with the RandomX team, RandomX is not intended to work on 32-bit legacy systems, as they will never be meaningfully competitive on a modern proof-of-work algorithm. Pull Request 99 adds a check to properly throw an error instead of silently failing.

Issue 4 – FIXED – Incorrect Code Generated in Emulation Mode – Pull Request 98
X41 found multiple issues with RandomX running in emulation mode with non-default parameters. This is important because the default parameters are only intended to be used with Monero, and other projects will be using non-default parameters. Pull request 98 corrects these issues.

Issue 5 – FIXED – Insufficient Diffusion in AesGenerator4R – Pull Request 76
X41 found that key-reuse in the AesGenerator4R function allows for trivial reversal of the function. It is a best practice to use individual keys in AesGenerator4R. It is important to note that in the specific way that RandomX uses the AesGenerator4R function, this does not present a security risk because the input is a cryptographically secure hash function.

Issue 6 – FIXED – Poor Code Coverage – Pull Request 73
X41 noted that RandomX is monolithic and that it is hard to test individual components of RandomX rather than RandomX as a whole. This is important because rare problems can be hidden within the components that don't get tested directly. In pull request 73, test vectors were added to the code as well as additional regression testing.

Issue 7 – FIXED OPTIONALLY – JIT Memory Pages for Generated Code are Writable and Executable – Pull Request 112
The JIT compiler has simultaneous read, write, and execute permissions which are not required. Removing the ability for the JIT compiler to write and execute at the same time mitigates the possibility of an attacker exploiting a bug in the JIT compiler to execute arbitrary code. This was fixed in pull request 112, however, the switching causes a large hit on mining performance (~30%) that will be unacceptable to most. Because of this, it has been made an option to enable/disable.

outside source (a block) and process it, it can't be ruled out that an attacker could find inputs that could bypass the security properties of the RandomX Virtual Machine. Sandboxing techniques such as those provided by AppContiner or seccomp would help mitigate any possibility of such a scenario arising. The RandomX team disagrees with this assessment, because the values that an attacker can modify (sending fake blocks) are immediately fed through Blake2 hashing, which means that the inputs cannot be effectively controlled to cause an issue if a vulnerability in RandomX existed.

Issue 9 – FIXED – Incorrect SuperScalarHash Latency when Program Size is too Small – Pull Request 98
Some combinations of parameters can cause invalid states where RandomX will run even though not enough memory is allocated for the application to function properly. Additional sanity checks for program size are recommended.

Issue 10 – Lack of a Machine-Readable Specification
RandomX doesn't have a machine readable spec to test against, meaning that there is no easy method to test if alternative implementations of RandomX will always, in all circumstances, have the same results. Having such a specification prevents issues with interoperability, and also assists developers of alternative implementations to detect and fix bugs in their own code. As the project matures, it is imperative that a machine-readable spec is established.

ASIC/FPGA Resistance 1 – DISPUTED – Low Reliance on Branch Prediction Unit
The branch prediction unit of a CPU is a complex and large component that takes up a lot of physical space on a CPU. RandomX does not rely on branch prediction logic, so an ASIC could greatly simplify a potential ASIC by simplifying the branch prediction unit. After talking with the RandomX team, this is disputed because any logic that can be predicted by a branch predictor can be implemented in an ASIC, possibly more efficiently.

ASIC/FPGA Resistance 2 – Low reliance on Front-End Scheduling Logic
Some parts of RandomX can be executed in parallel by using a preprocessor instead of the JIT, allowing a custom-chip design to have an advantage over CPUs. After talking with the RandomX team, this area of the code is not high-impact and does not

ASIC/FPGA Resistance 3 – DISPUTED – Low Reliance on Other Modern CPU Elements
RandomX does not use many complex modern CPU features, such as: privileged instructions, interrupts and interrupt routing (x86 APIC), TLB 8 s, the MMU 9 and virtual memory in general, inter-core communication and cache coherence. While these features do not take up a lot of die space, they are very complex and require large amounts of R&D to implement. Adding features to RandomX that use these CPU features would dramatically increase the cost of a custom chip. RandomX cannot implement many custom features because it is designed to work on many different CPU architectures that may lack those features.

Audit 4 – QuarksLab sponsored by OSTIF, Monero Research Lab, and the Monero Community

This audit was conducted by QuarksLab after the conclusion and publishing of the other three audits, to maximize the impact of their review, they considered the latest build of RandomX and all of the work of the previous teams.

Issue 1 – (RNDX-N1) Usage of Hard Coded Strings
RandomX made significant effort to make components of its protocol fully customizable for a safe reuse. There are six hard coded strings as seeds of various constants and keys and it's suggested to add these strings in the configuration set for easier management and customization. *This is a minor code practices issue and not a security vulnerability.*

Issue 2 – FIXED – (RNDX-N2) Warning Against RandomX Components Reuse – Pull Request 111
While the existing implementation of the components AesHash1R, AesGenerator1R, and AesGenerator4R are sound, there should be warnings about the re-use of these functions for developers who may be tempted to use these functions elsewhere in their own

Issue 3 – FIXED – (RNDX-M1) Argon2 Implementation Does Not Enforce Minimum Salt Size – Pull Request 111
Argon2 requires a salt of a minimum size of 8 bytes. RandomX does not enforce this and does not throw an error if the salt is too small. A lower-bound sanity check should be added.

Issue 4 – FIXED – (RNDX-L1) RANDOMX_ARGON_LANES and RANDOMX_ARGON_ITERATIONS Need Upper Bound Limits – Pull Request 111
RANDOMX_ARGON_LANES needs an upper-bound limit of 2^24-1 and RANDOMX_ARGON_ITERATIONS needs an upper-bound limit of 2^32-1.

Issue 5 – FIXED – (RNDX-L2) RANDOMX_ARGON_MEMORY Needs A Lower Bound Limit of 8 – Pull Request 111
It has a lower-bound limit, but it is currently 1 which doesn't consider that ARGON2_SYNC_POINTS = 4.

Issue 6 – FIXED – (RNDX-L3) Missing 64-byte Alignment of mx and ma in the Specifications – Pull Request 111

Issue 7 – FIXED – (RNDX-M2) Error in Computation of DataOffset
The calculations for DataOffset did not match the specification. After QuarksLab discussed this with the RandomX team, it was discovered that this was a snippet of legacy code from an older implementation of RandomX. This issue has been corrected.

Issue 8 – FIXED – (RNDX-M3) RandomX Key Size Limit Not Enforced – Pull Request 111
RandomX specifies a 60-byte input key, but larger keys can be loaded without error. After talking with the RandomX team, larger keys are truncated to 60 bytes by design. RandomX now informs users when longer keys have been truncated.

The Audit Papers

Trail of Bits
Kudelski Security
X41 D-Sec
QuarksLab

social media (@ostifofficial on Twitter, /u/ostifofficial on Reddit) and consider spreading the word about our cause. We work hard on these projects and none of this work gets done without funding!

← Announcing the OSTIF Anti-Censorship Project

OSTIF is Hosting a KickStarter to Raise Money for Open Source Projects! →

Copyright © 2020 Open Source Technology Improvement Fund. All rights reserved.
Theme: ColorMag by ThemeGrill. Powered by WordPress.