SECURING EVERY BIT OF YOUR DATA

Quote MD/2019/05/572-PRO

Quote date : 10-mai-19

Quarkslab 13 rue St Ambroise, 75011 Paris, France Tél. : +33 6 33 28 11 77 Email : mduez@quarkslab.com	The Monero Lab

Reference	Description	Days	Price per day	Cost
19-04-563-PRO	**Core Dalek libraries review**			
	Step 1 : Global understanding specification and components overview	5	$ 1 650,00	$ 8 250,00
	Step 2: Checking that the RandomX specification is cryptographically secured and doesn't allow algorithmic optimizations	10	$ 1 650,00	$ 16 500,00
	Step 3: Validating the code matches the specification + vulnerability analysis	10	$ 1 650,00	$ 16 500,00
	Step 4: Verifying that implementation is "optimization free"	7	$ 1 650,00	$ 11 550,00
	50% at the launch of the mission 50% upon completion of the mission			
			Total $ excl VAT	$ 52 800,00

Online payment max 30 days after bill
Late payment penalties : BCE rate + 10 points

Quarkslab, SAS au capital de 113 390 €
RCS : : Paris 538 485 897
SIRET : 538 485 897 00036
APE : 7490B
TVA FR : FR 12 538 485 897 00036

Payment to :
Banque : Banque Palatine
RIB :40978 00022 1384828V800 25
IBAN : FR74 4097 8000 2213 8482 8V80 025
BIC :BSPFFRPPXXX

STATEMENT OF WORK

Description of the request
==========

The Monero developer community has expressed interest in the RandomX proof-of-work algorithm. They want to realize a code review / audit of the algorithm to ensure that :
- implementation of the protocol is well respected,
- there are no vulnerabilities,
- criterias for a proof of work are met.

Criteria for a proof of work algorithm are :
1) Optimization-free: There is no algorithmic speed-up that allows one to calculate the hash faster than the reference algorithm.
2) Progress-free: Proof of work calculation doesn't depend on the history of previous calculations.
3) Approximation-free: It's not possible to achieve speed-up larger than the inverse rate of invalid hashes

Monero Lab expects a fully audit (implementation review and vulnerability analysis) of the basic implementation (hashing function itself which includes VM implementation).
For the components used to speed up things (JIT,large pages etc.) and for the 3rd-party components (Blake, AES, Argon), the audit only verifies that they are correctly manipulated by randomx (lack of vulnerability). The analysis of components itself is out of scope.
For "optimization free" criteria, the audit verifies that there is no other way for ASIC but to fully implement this VM and that there are no ways to make it run much faster on CPU than with current JIT code (around 4000-5000 lines of code to audit).

Quarkslab's work description
==========

The evaluation that Quarkslab will undertake includes the four following steps:
1- Global understanding specification and RandomX components overview
2- Checking that the RandomX specification is cryptographically secured and doesn't allow algorithmic optimizations
3- Validating that the code matches the specification + vulnerability analysis
4- Verifying that implementation is "optimization free"

At the end of the evaluation, Quarkslab will produce a report detailing the evaluation activities undertaken and results achieved.
Considering the novelty and difficulty of the task, we are proposing a team of 2 senior evaluators in cryptography, implementation of cryptography and vulnerability research.
The daily rate for this type of audit is 2000$/day that is reduced to 1650$/day in case report can be communicated and published in coordination with Monero Lab.

Detailed breakdown of the source tree

==========

asm, blake2 > vulnerability analysis only
tests -> can be skipped
aes_hash.cpp 137
aes_hash.hpp 29
argon* -> vulnerability analysis only
assembly_generator* -> can be skipped
allocator.cpp 52
allocator.hpp 37
blake2_generator.cpp 53
blake2_generator.hpp 37
common.hpp 156
configuration.h 112
dataset.cpp 170
dataset.hpp 77
instruction.cpp 94 (printing removed)
instruction.hpp 140
instruction_weights.hpp 105
instructions_portable.cpp 163
intrin_portable.h 322
jit_compiler -> vulnerability analysis only
program.hpp 63
randomx.cpp 241
randomx.h 189
reciprocal.c 59
reciprocal.h 33
soft_aes* -> vulnerability analysis only
superscalar.cpp 888
superscalar.hpp 51
superscalar_program.hpp 71
virtual_machine.cpp 128
virtual_machine.hpp 67
virtual_memory.cpp 121
virtual_memory.hpp 27
vm_compiled* -> vulnerability analysis only
vm_interpreted.cpp 663
vm_interpreted.hpp 90
vm_interpreted_light.cpp 47
vm_interpreted_light.hpp 52

~4500 lines of code in total