Arweave 2.9 Upgrade Security Assessment

Forward Research
Version 1.0 – December 6, 2024

Prepared By
Gerald Doussot
Parnian Alimi

Prepared For
Sam Williams
James Piechota

1 Executive Summary

Synopsis

During December of 2024, Forward Research engaged NCC Group's Cryptography Services team to conduct a security assessment of the Arweave 2.9 upgrade. This network upgrade introduces a much more efficient entropy generation and data preparation mechanism which will considerably decrease honest miners' computation cost, and in turn further incentivizes wider adoption. Two consultants delivered this engagement in 4 person-days.

A detailed description of the scope and the proposal was provided at the kick-off. The Arweave team answered a number of questions asked by NCC Group consultants in Slack.

Scope

The scope was limited to the implementation of the *RandomXSquared* algorithm in Pull Request #724 on commit `09d3fdb`. The assessment was assisted by the fourth draft of the "Efficient generation and dispersion of entropy during data preparation for Arweave" document. The main Arweave protocol (https://draft-17.arweave.dev/) was used as a reference for various aspects of the protocol.

The *RandomX* function itself was out of scope.

Limitations

NCC Group was able to achieve reasonable coverage of the in scope components in the time allocated to the project. Note, however, that the proposed data preparation mechanism is part of a bigger system which should be studied separately.

Key Findings

The assessment uncovered two findings.

- Finding "Computation of Different CRC32 Values Depending On Architecture" notes that computing different CRC32 values across different CPU architectures may result in interoperability issues, and at least transient unavailability of data if the issue is not fixed.
- Finding "Potential Memory Leak in Failure Paths" warns that failure to properly release memory on failure paths may affect host's performance.

Further discussion of the protocol is included in the Protocol Specification Review section.

Strategic Recommendations

After addressing the findings presented in this review, NCC Group recommends running integration tests to ensure interoperability across architectures.

2 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Computation of Different CRC32 Values Depending On Architecture	New	2ND	Medium
Potential Memory Leak in Failure Paths	New	GRJ	Low

3 Finding Details

Medium

Computation of Different CRC32 Values Depending On Architecture

Overall Risk	Medium	**Finding ID**	NCC-E020578-2ND
Impact	High	**Category**	Cryptography
Exploitability	Medium	**Status**	New

Impact

Computing different CRC32 values across different CPU architectures may result in interoperability issues, and at least transient unavailability of data if the issue is not fixed.

Description

Arweave employs packing to ensures miners cannot forge Succinct Proof of Access (SPoA) proofs claiming to represent multiple unique replicas from the same stored data chunk. In Arweave 2.8, the chunk offset, transaction root and mining address are combined in a SHA-256 hash to generate a packing key. The packing key is used as entropy for an algorithm that generates a scratchpad over several successive rounds of the `RandomX` primitive execution. The resulting entropy is used to symmetrically encrypt the chunk to produce a packed chunk and store it. In Arweave 2.9, a number of changes were implemented. Notably for this finding, `RandomX` execution is interleaved with calls to the CRC32 and deterministic Fisher–Yates shuffle primitives on the scratchpad data with the goal of facilitating full use of the scratchpad entropy for honest miners.

Function `packing_mix_entropy_crc32()` in file pack_randomx_square.cpp calls C macro `crc32()` to compute the CRC32 checksum on the data 32 bits at a time:

```
void packing_mix_entropy_crc32(
      const unsigned char *inEntropy,
      unsigned char *outEntropy, const size_t entropySize) {
// SNIP
    const unsigned int *inEntropyPtr = (const unsigned int*)inEntropy;
    unsigned int *outEntropyPtr = (unsigned int*)outEntropy;
    for(size_t i=0;i<entropySize;i+=4) {
// SNIP
      state = ~crc32(state, *inEntropyPtr);
      *outEntropyPtr = *inEntropyPtr ^ state;
      inEntropyPtr++;
      outEntropyPtr++;
    }
// SNIP
    // Note it's optimizeable now with only 1 pointer, but we will reduce readability later
    for(size_t i=0;i<entropySize;i+=4) {
      //state = crc32(~state, *inEntropyPtr);
      //*outEntropyPtr = *inEntropyPtr ^ ~state;

      state = ~crc32(state, *inEntropyPtr);
      *outEntropyPtr = *inEntropyPtr ^ state;
      inEntropyPtr++;
      outEntropyPtr++;
    }
  }
```

This macro, defined in file crc32.h, wraps compiler intrinsics implementing the checksum algorithm for Intel and ARM architectures, as illustrated below:

```
#ifndef CRC32_H
#define CRC32_H

#if defined(__x86_64__) || defined(__i386__) || defined(_M_X64) || defined(_M_IX86)
  #include <immintrin.h>
  #define crc32(a, b) _mm_crc32_u32(a, b)
#elif defined(__aarch64__) || defined(__arm__) || defined(_M_ARM64) || defined(_M_ARM)
  #include <arm_acle.h>
  #define crc32(a, b) __crc32d(a, b)
#else
  // TODO make support for soft crc32
  #error "Unsupported architecture for CRC32 operations."
#endif

#endif // CRC32_H
```

On Intel, the `_mm_crc32_u32()` function accepts two 32 bit unsigned integers and returns a 32 bit unsigned integer. On ARM the `__crc32d()` function accepts one 32 bit unsigned and a 64 bit unsigned integer, and returns a 32 bit unsigned integer. In function `packing_mix_entropy_crc32()`, the macro parameter `inEntropyPtr` is a 32-bit value, so it is implicitly extended to 64 bits in the ARM case, thus computing a different value than on Intel. Furthermore CRC32 is parameterized by the primitive polynomial in GF(2)[X]] of degree 32. The Intel instruction implements only one polynomial. On ARM two distinct polynomials are implemented, and the right one is chosen by the value of a specific bit in the instruction encoding. The use of `__crc32cw()` sets this bit and it then selects the same polynomial as Intel. The `__crc32d()` uses a different polynomial than on Intel, and would therefore return the wrong value, even if the operand where the same size.

Recommendation

Change the ARM intrinsics from `__crc32d()` to `__crc32cw()`. Ensure integration testing is performed across different architectures.

Reproduction Steps

Macro `crc32()` in file crc32.h

Location

https://github.com/ArweaveTeam/arweave-dev/blob/02ece0ff34761b2071641b38dd62b6d4a43e55f9/apps/arweave/c_src/randomx/crc32.h

Low

Potential Memory Leak in Failure Paths

Overall Risk	Low	**Finding ID**	NCC-E020578-GRJ
Impact	Low	**Category**	Session Management
Exploitability	Undetermined	**Status**	New

Impact

Failure to release allocated memory may affect the host's performance. This can be fatal in embedded systems.

Description

In the following code snippet from the `rsp_exec_nif()` implementation, after input parameters are parsed, a *RandomX* Virtual Machine (VM) is created:

```
static ERL_NIF_TERM rsp_exec_nif(
    ErlNifEnv* envPtr, int argc, const ERL_NIF_TERM argv[]) {
  // ... snip...
  int isRandomxReleased;
  randomx_vm *vmPtr = create_vm(statePtr, (statePtr->mode == HASHING_MODE_FAST),
      jitEnabled, largePagesEnabled, hardwareAESEnabled, &isRandomxReleased);
  if (vmPtr == NULL) {
    if (isRandomxReleased != 0) {
      return error_tuple(envPtr, "state has been released");
    }
    return error_tuple(envPtr, "randomx_create_vm failed");
  }

  outScratchpadData = enif_make_new_binary(
    envPtr, randomx_get_scratchpad_size(), &outScratchpadTerm);
  if (outScratchpadData == NULL) {
    return enif_make_badarg(envPtr);
  }

  outHashData = enif_make_new_binary(envPtr, 64, &outHashTerm);
  if (outHashData == NULL) {
    return enif_make_badarg(envPtr);
  }

  randomx_squared_exec(
    vmPtr, inHashBin.data, inScratchpadBin.data, outHashData, outScratchpadData,
    randomxProgramCount);

  destroy_vm(statePtr, vmPtr);

  return ok_tuple2(envPtr, outHashTerm, outScratchpadTerm);
}
```

Once a VM is successfully created, it should be destroyed if the program exits for any reason (including failure to allocate memory) by calling `destroy_vm()`. The severity of this finding is low, since modern hosts will usually reclaim a process's memory after it exits. This note also applies to the `rsp_init_scratchpad_nif()` implementation.

Recommendation

Call `destroy_vm()` before the function returns in the 2 highlighted lines above.

Location

- https://github.com/ArweaveTeam/arweave-dev/blob/feature/fast-replication/apps/arweave/c_src/randomx/rxsquared/ar_rxsquared_nif.c#L96
- https://github.com/ArweaveTeam/arweave-dev/blob/feature/fast-replication/apps/arweave/c_src/randomx/rxsquared/ar_rxsquared_nif.c#L101
- https://github.com/ArweaveTeam/arweave-dev/blob/feature/fast-replication/apps/arweave/c_src/randomx/rxsquared/ar_rxsquared_nif.c#L160
- https://github.com/ArweaveTeam/arweave-dev/blob/feature/fast-replication/apps/arweave/c_src/randomx/rxsquared/ar_rxsquared_nif.c#L165

4 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Rating	Description
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

5 Protocol Specification Review

The following material discusses various aspects of the suggested optimizations in the 2.9 upgrade, as is described in the “Arweave_2-9-draft-4.pdf”.

Notes on the mixing algorithm

One major improvement of the 2.9 upgrade is the introduction of the *RandomXSquared* algorithm. By running multiple RandomX entropy generators in parallel lanes, mixing the resulting entropy, and then repeating this process many rounds (a_m), *RandomXSquared* produces the sufficient amount of entropy to pack an entire zone in the weave. The mixing step (a_{gm}) after every round is specified as CRC32 followed by Fisher-Yates shuffling. The shuffling is necessary to prevent on-demand packing, i.e., without shuffling a dishonest miner could only execute the lane that corresponds to their chunk and truncate the amount of necessary work. Although the proposal envisages a random Fisher-Yates shuffle, it suffices to run a deterministic shuffler that mixes entropy from all lanes to initialize a given lane’s scratchpad for the next round. In fact the implementation utilizes a simple mechanism to mix the entropy. As depicted below blocks at index `offset` within every `jumpSize` interval are copied next to each other, in the order they appear in lanes:

```
void packing_mix_entropy_far(
    const unsigned char *inEntropy,
    unsigned char *outEntropy, const size_t entropySize,
    const size_t jumpSize, const size_t blockSize) {
  unsigned char *outEntropyPtr = outEntropy;
  size_t numJumps = entropySize / jumpSize;
  size_t numBlocksPerJump = jumpSize / blockSize;

  for (size_t offset = 0; offset < numBlocksPerJump; ++offset) {
      for (size_t i = 0; i < numJumps; ++i) {
         size_t srcPos = i * jumpSize + offset * blockSize;
         memcpy(outEntropyPtr, &inEntropy[srcPos], blockSize);
         outEntropyPtr += blockSize;
      }
  }
}
```

NCC Group’s analysis of this mixing algorithm concluded that it is as effective as a Fisher-Yates shuffle in preventing work truncation. Additionally, this algorithm simplifies the mining verifiers’ work to produce the same entropy, unpack chunks, and validate proofs.

As an aside, the implementation uses 128 lanes (a_l), 4 depths (a_m), 1024 bytes block sizes, and jump sizes equal to the RandomX scratchpad size.

Notes on the RandomX program size

RandomX’s VM execution, executes a program iteratively (controlled by the `RANDOMX_PROGRAM_ITERATIONS` parameter, and set to 2048 by default). The *RandomXSquared* algorithm breaks these iterations among each row. A single *RandomX* execution within *RandomXSquared*, iterates 512 times, with 4 rows produces the same number of program iterations (2048) for every lane.

Note that Algorithm 1 in the paper, correctly, sets the $RX2_{progs}$ to $2048/a_m$, but Algorithm 2 on line 8 implies that *RandomXSquared* uses the same program iterations as RX_{progs}, which is incorrect.

6 Contact Info

The team from NCC Group has the following primary members:

- Gerald Doussot – Consultant
  gerald.doussot@nccgroup.com
- Parnian Alimi – Consultant
  parnian.alimi@nccgroup.com
- Javed Samuel – Cryptography Services Director
  javed.samuel@nccgroup.com

The team from Forward Research has the following primary member:

- Sam Williams
  sam@arweave.org