Cloud Software Group

NetScaler CPU Jitter Entropy Source

v3.4.0

SP 800-90B Non-Proprietary Public Use Document CPU Jitter (JENT) v3.4.0

Document Version: 0.2

Prepared for:

Cloud Software Group
851 Cypress Creek Road
Fort Lauderdale, FL 33309
United States of America

Phone: +1 954 267 3000
www.cloud.com

Prepared by:

Corsec Security, Inc.
12600 Fair Lakes Circle, Suite 210
Fairfax, VA 22033
United States of America

Phone: +1 703 267 6050
www.corsec.com

Revision History

Version	Code Version	Modification Date	Modified By	Description of Changes
0.1	3.4.0	2023-02-27	Steele Myrick	Initial draft.
0.2	3.4.0	2023-04-13	Steele Myrick	Corrected JENT code version to 3.4.0

Table of Contents

1. Description **4**
2. Security Boundary **5**
3. Operating Conditions **6**
4. Configuration Settings **7**
5. Physical Security Mechanisms **8**
6. Min-Entropy Rate **9**
7. Health Tests **10**
8. Maintenance **11**
9. Required Testing **12**

List of Tables

Table 1 – Cloud NetScaler CPUs 4
Table 2 – Operating Conditions 6
Table 3 – Configuration Settings 7
Table 4 – Final Raw Min-Entropy Estimates 12
Table 5 – Final Restart Min-Entropy Estimates 12

List of Figures

Figure 1 – Entropy collection operation 5

1. Description

CPU Jitter Entropy (JENT) is a non-physical true random number generator source of entropy. It makes no IID claim and thus meets all the requirements for non-IID compliance. The report is for JENT version 3.4.0. The table summarizes the Cloud Software Group NetScaler CPU Jitter Entropy Source MPX and VPX platforms and their respective CPUs for which testing was performed.

Table 1 – Cloud NetScaler CPUs

Model	CPU
VPX on VMWare ESXi 7	Intel Xeon E5-2680 v4 (Broadwell)
MPX 89xx FIPS	Intel Xeon E5-2620 v4 (Broadwell)
MPX 15xxx-50G FIPS	Intel Xeon E5-2620 v4 (Broadwell)
MPX 91xx FIPS	Intel Xeon Silver 4310T (Ice Lake)

2. Security Boundary

The boundary of the JENT implementation is shown in Figure 1. The basic concept that is implemented with the CPU Time Jitter NPTRNG can be summarized as follows: The unpredictable phenomenon of variances in the execution time of a given set of instructions is collected and accumulated. This set of instructions is separated into the SHA3-256 component and the memory access component. The measurement of the execution time jitter is performed over the post-processing logic of the SHA3-256 and supportive functions, i.e., the CPU Time Jitter NPTRNG measures the execution time of the SHA3- 256 and the execution time of memory access. After which the execution time is injected into the SHA3-maintained entropy pool. The JENT output is used seed an SP 800-90A compliant DRBG (which is outside of the boundary of the entropy source). The security boundary of the JENT implementation contains the source code files provided as part of the software delivery. The source code contains API calls which are used by modules requesting entropy from JENT.

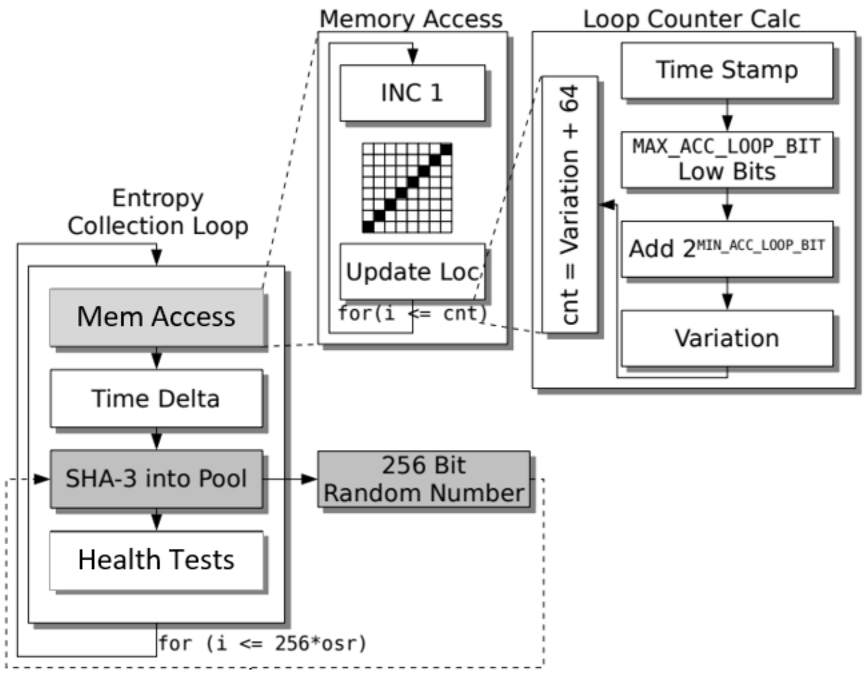

Figure 1 – Entropy collection operation

3. Operating Conditions

Table 2 summarizes the operating conditions for each of the tested CPUs.

Table 2 – Operating Conditions

CPU	Operating System	Platform	Platform Temperature	Platform Voltage (AC)
Intel Xeon E5-2680 v4, Broadwell, 2.4 GHz	FreeBSD 11.4 on VMware ESXi 7	VPX	NA	NA
Intel Xeon E5-2620 v4, Broadwell, 2.1 GHz	FreeBSD 11.4	MPX 89xx FIPS	10°C – 80°C	100V – 240V
Intel Xeon E5-2620 v4, Broadwell, 2.1 GHz	FreeBSD 11.4	MPX 15xxx-50G FIPS	0°C – 80°C	100V – 240V
Intel Xeon Silver 4310T, Ice Lake, 2.1 GHz	FreeBSD 11.4	MPX 91xx FIPS	10°C – 80°C	100V – 240V

4. Configuration Settings

Table 3 summarizes the configuration settings used by the Cloud implementation of JENT. These settings must be preserved to comply with the JENT ESV certificate.

Table 3 – Configuration Settings

Setting	Value	Description
JENT_RANDOM_MEMACCESS	Enabled	Determines the method of memory access
JENT_CONF_ENABLE_INTERNAL_TIMER	Disabled	Disables the use of internal timers
JENT_CONF_DISABLE_LOOP_SHUFFLE	Enabled	Disables the random number of hash loops performed for each call (only one hash operation is performed with this setting)
JENT_MIN_OSR	3	Oversampling rate

5. Physical Security Mechanisms

The JENT implementation is a software module. As such, it does not include physical security mechanisms. The JENT module may be used on a variety of devices with differing security levels. Physical security requirements are, therefore, dependent on the modules that use JENT.

Conceptual Interfaces provided by JENT:

- GetEntropy: The call to jent_read_entropy serves as a GetEntropy interface according to the SP 800-90B definition. The jent_read_entropy function generates a random number in a way that is compliant with all SP 800-90B requirements. The caller specifies the size of the random value that it wants the function to return. The function sets a return code to indicate whether or not the request was fulfilled successfully. The return code also reports failures of the health tests.

- GetNoise: The JENT library does not have a GetNoise interface, but JENT comes with scripts that collect the raw noise data (jitterentropy-rng). The scripts may be used to compile specific executables that collect 1,000,000 consecutive samples or 1,000 sets of 1,000 samples of raw data.

- HealthTest: The JENT library does not have a HealthTest interface, but health tests can be run by allocating a new JENT handle. The health tests are compliant with SP 800-90B.

6. Min-Entropy Rate

The JENT implementation generates an output that is considered to have full entropy. A request for 256 bits of entropy results in 256 bits of entropy per output sample, or full entropy.

7. Health Tests

Per the SP 800-90B requirements, health tests run when JENT starts, and are then run continuously while it is operating. All tests check for persistent failures base on their respective “cutoff” values, which represent expected error thresholds.

JENT implements four types of health tests:

- Stuck Test
- Repetition Count Test (RCT)
- Adaptive Proportion Test (APT)
- Lag Predictor Test

The stuck test calculates the first, second, and third discrete derivative of the time to be processed by the hash. The received time delta is considered to be non-stuck only if all three values are non-zero.

The Lag Predictor Test is a vendor-defined conditional test that is designed to detect a known failure mode where the result becomes mostly deterministic. It is based on the Lag Prediction Test described in SP 800-90B, section 6.3.8.

The RCT and APT tests are implemented as described in SP 800-90B.

When any health test fails, the API call to generate random numbers (jent_read_entropy(3)) informs the caller about the failure with error codes and the Jitter RNG block causing the failure is not returned to the caller.

All health test failures are considered permanent failures. If one is triggered, the current instance of the Jitter RNG will always remain in error state. The documentation of the API call for jent_read_entropy(3) explains that the caller can only clear this error state by deallocating the Jitter RNG instance followed by an allocation of a new Jitter RNG instance to reset the noise source.

8. Maintenance

There are no maintenance requirements for the JENT library.

9. Required Testing

Raw data was collected by running the invoke_testing.sh script included in the JENT package. This script first gathers 1,000,000 consecutive samples of raw data. It then gathers 1,000,000 samples of raw data in groups of 1,000 samples and restarting JENT after each group of data for the restart tests.

For best compliance to SP 800-90B, the number of hashing operations is always one (and is the default setting for JENT 3.4.0).

Raw Data Analysis

Each raw data sample consists of one timestamp delta, which is 64 bits long. It is assumed that only the least significant 4 bits of each timestamp delta contains any true entropy. The JENT design states that the Jitter RNG can deliver full entropy if and only if the min-entropy is at least $\frac{1}{osr}$ bit of entropy per timestamp. The Jitter RNG implementation uses an oversampling rate (OSR) of 3.

Table 4 below shows that the min-entropy for the raw data samples ranges from 3.446767 to 3.877634 bits of entropy per 4-bit sample, thus greatly exceeding the requirement to claim full entropy.

Table 4 – Final Raw Min-Entropy Estimates

Entropy Result	VPX	MPX 8900	MPX 9100	MPX 15000-50G
H_original	3.687754	3.687754	3.877634	3.687754
H_bitstring	0.861692	0.884718	0.916641	0.884718
min(H_original, 4 X H_bitstring)	3.446767	3.538870	3.666563	3.538870

Restart Data Analysis

The CPU Jitter RNG does not have a stochastic model, but an $H_{submitter}$ estimate of 1 was determined using analysis as described in SP 800-90B Section 3.2.2, Requirement 3. The Jitter RNG implementation uses an oversampling rate of 3. For that reason, $H_{submitter}$ is set to 0.333 ($\frac{1}{3}$) for the purpose of running the NIST restart tests.

For the restart tests, the raw entropy data is collected for 1,000 Jitter RNG instances allocated sequentially. That means, for one collection of raw entropy, one Jitter RNG instance is allocated. After the conclusion of the data gathering it is deallocated and a new Jitter RNG instance is allocated for the next restart test round.

Table 5 – Final Restart Min-Entropy Estimates

Entropy Result	VPX	MPX 8900	MPX 9100	MPX 15000-50G
H_r	3.882348	3.872578	3.687754	3.872578
H_c	3.687754	3.773078	3.891917	3.773078
H_I	0.333333	0.333333	0.333333	0.333333
min(H_r, H_c, H_I)	0.333333	0.333333	0.333333	0.333333