# Entropy is costing time and money.

# Why shouldn’t you care?

Alan Baugher, Sr. Principal Architect

**March, 2017   (Refresh)**

ca technologies

# Entropy Quick Test on Unix/Linux Servers

The Entropy range is from **0 - 4096**

Execute this command on your Linux/Unix platforms:

**watch -n 1 cat /proc/sys/kernel/random/entropy_avail**

yum –y install rng-tools

If the return value is less than 1000, then please think about adding an Entropy Pump to all of your server(s).

Open another Putty Window, then execute this command to emulate a password command:

**time dd if=/dev/random bs=8 count=1 2> /dev/null | base64**

Want to know what processes are using Entropy? Execute this command:

**lsof | grep -E "/dev/[u]{0,1}random"** {assumes that lsof is installed & user has access to run this command}

*** Please do NOT use the "software hack" of replacing /dev/random with a soft link from /dev/urandom ***
*** The OS, upon any update, may rebuild the device driver of /dev/random ; /dev/random is a BLOCKING device driver ***




technologies

# Entropy Impacts to Business & Infrastructure

- Business Concerns:
  - User Experience
  - Reliability
  - Productivity decreased while waiting for systems.
  - Cost
    - Over-purchase of assets to address perceived low performance with existing environment infrastructure
    - Resources (Internal/Vendor/Consultants) waiting on cycle of solutions for testing or production use
  - Audit Impact of FIPS-140-2 functionality.

- Technical Concerns:
  - Performance for ALL JVM (Weblogic / JBOSS / WebSphere)
    - Startup time is increased
    - LDAP/S binds duration is increased.
  - Performance & Install for CA IM JCS (IAMCS) with FIPS
    - Connections to endpoints with SSL/TLS security is increased
    - Impact to LDAP/S and JDBC/S
    - Install will fail if FIPS is enabled
  - Performance for CA SSO/Siteminder
    - LDAP/S binds may take over 90+ seconds instead of < 1 second
    - Startup time is increased.
    - Installation time is increased.
  - Any SSL/TLS/Certs Generation solutions on UNIX/Linux - Including Any Directory or Database solution.

# BUSINESS HIGH LEVEL SUMMARY

- Entropy is used to secure the Cloud and Servers
  - It is used for all eCommerce sites, financial sites, etc.
  - It is ubiquitous for ALL security solutions
  - It is used in ALL J2EE platforms (Oracle Weblogic, IBM Websphere, RHEL JBOSS)
  - It is used for HTTPS, JDBC(S), LDAPS, etc. protocols.

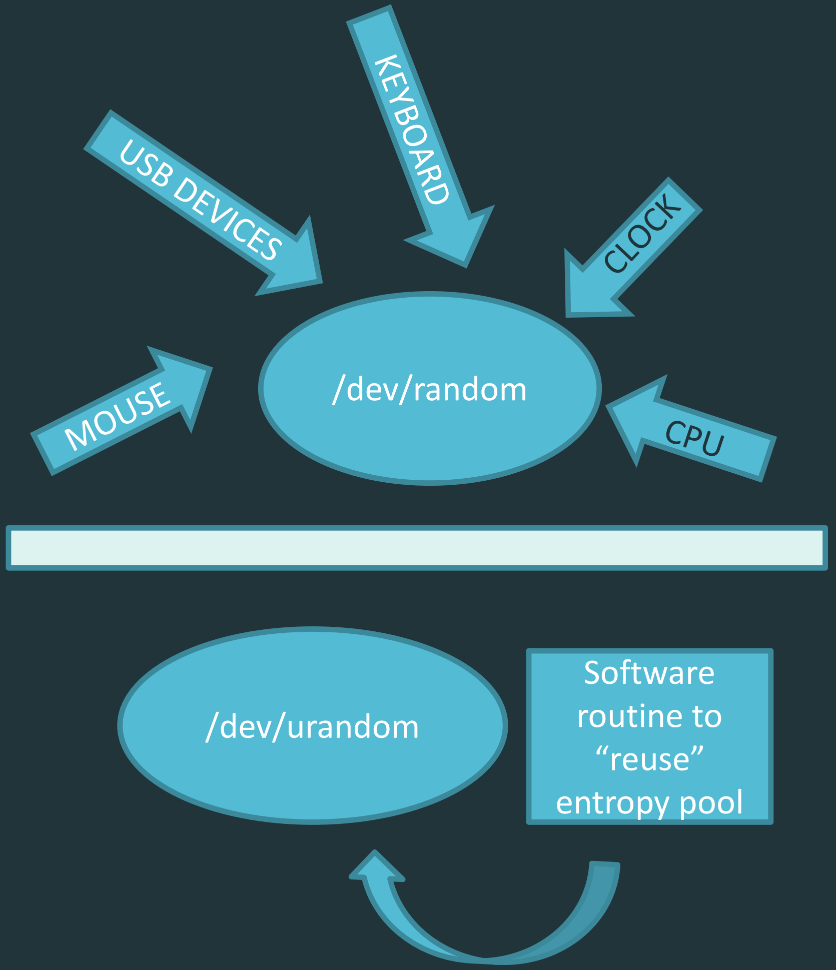
- Compliance
  - Is Your Entropy "random" enough?
    - http://www.forbes.com/2009/07/30/cloud-computing-security-technology-cio-network-cloud-computing.html
    - https://www.schneier.com/blog/archives/2013/10/insecurities_in.html

- Cost
  - Is Entropy making you **over purchase** hardware for perceived performance challenges?
    - The very design of Entropy on Linux/UNIX platforms, with default settings, impacts startup and ongoing processes that appear to be "slow" or "halt".
  - Do NOT accept default settings
    - Require the IT team investigate use of entropy "pumps" to drive performance before the next investment of H/W for performance issues.

# TECHNICAL HIGH LEVEL SUMMARY

- Background:
  - OS vendors have configured OS packages (OpenSSL/SSHD/Libcrypt) to default to **/dev/urandom** for FIPS-140-2 and non-FIPS certification.
  - Most non-OS vendor cryptography software is hard coded or default configured to use **/dev/random** on Linux/UNIX OS
  - /dev/random is passive and a "blocking" device driver
- Recommendation:
  - Do **NOT** follow advice that destroy the trust relationship between /dev/random and /dev/urandom with softlinks [OS will likely rebuild them away; and you will be back to original challenge]
  - Use a "pump" to push in entropy to /dev/random "well". [which will "feed" /dev/urandom]
  - Pick an acceptable source for the "pump" to maintain FIPS-140-2 certification. [Acceptable sources are hardware inducing "entropy"]
- Validation:
  - Monitor startup times before and after using a "pump".

- Get the performance expected from your virtualized environment!



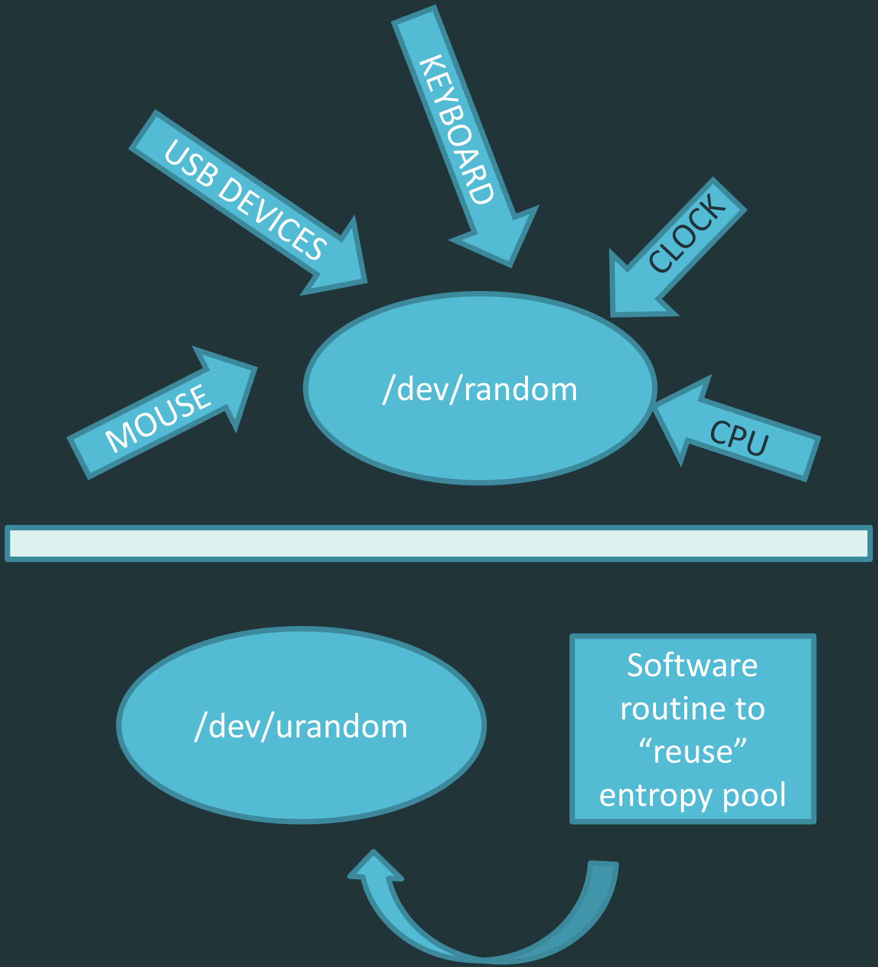

## What is Entropy / Entropy Pool

- An Entropy Pool is used by cryptography routines within software, e.g. SSL, TLS, Certificates, encryption software, etc.
-Two device drivers exist on UNIX/Linux OSes to support use of an entropy pool.

- **/dev/random** collects it's randomness via environment (physical input via mouse, CPU, Keyboard, Clock, USB devices, etc.). This is the default driver used by cryptography software (direct/indirect), to ensure a high level of confidence in the security of the output.

-**/dev/urandom** collect it's randomness by reusing the existing "entropy pool", which provides pseudo-random numbers.

-Many OS vendors have passed FIPS-140-2 certifications for OpenSSL/SSHD/Libcrypt using /dev/urandom; as long as the underlying trust relationship between /dev/random and /dev/urandom is not disturbed.



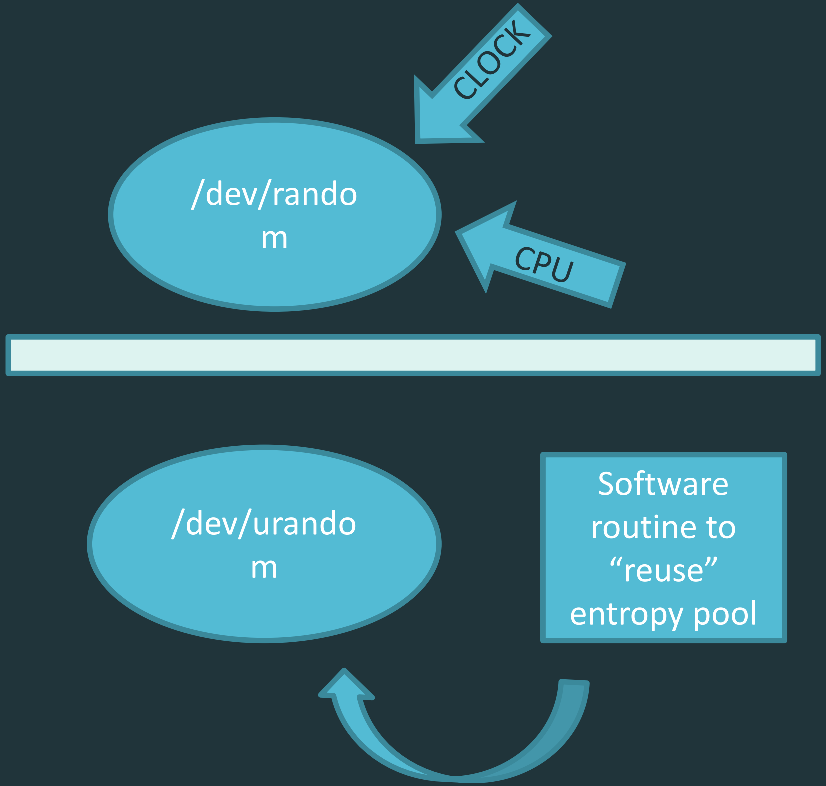

# What are these device drivers?

- **/dev/random** is used by default for all cryptography routines (direct/indirect) and is a "**blocking**" device driver.

**Challenge:** Intensive encryption processes may deplete the entropy pool of /dev/random and appear to "**halt**" software until the entropy pool is filled by additional environmental randomness.

This is observable during installation of certain encryption solutions where no cpu/no memory is used; until entropy pool number is sufficient for an installation to continue. /dev/random is consider a "**blocking**" pool for this reason, to ensure security.

-**/dev/urandom** is a "**non-blocking**" device driver that is constantly refreshed by the reuse of the entropy pools to provide pseudo-random numbers. This device driver **never** gets depleted.




# Why is this a concern or issues?

## Greater Challenge for virtual / headless servers:

The environmental devices used to populate /**dev/random** are reduced and may not be configured to refresh the population of randomness to **/dev/random.**

When the entropy pool is depleted, software with cryptography routines are severely impacted with regards to performance, e.g. start-up duration of software, bind durations, generation of certificates, etc.

**Example:**
J2EE (Oracle Weblogic/IBM WebSphere/RHEL JBOSS) startup may take > 15minutes instead of < 5minutes.

Web Access Control solutions bind to LDAP/S user store may take 90+ seconds to complete.

-**/dev/urandom** is constantly refreshed by the reuse of the entropy pools. This device driver **never** gets depleted, even for "virtual/headless" servers.

# How to address this entropy challenge on virtualized / headless servers

**Add a "pump" to refresh the "Entropy" well.**

**OS "Pump" Examples**

-RNGD

-HAVEGED





ca technologies

# OS RNGD Daemon

The OS RNGD (random number generator daemon) was introduced soon after entropy pools were created.

```
yum –y install   rng-tools
```

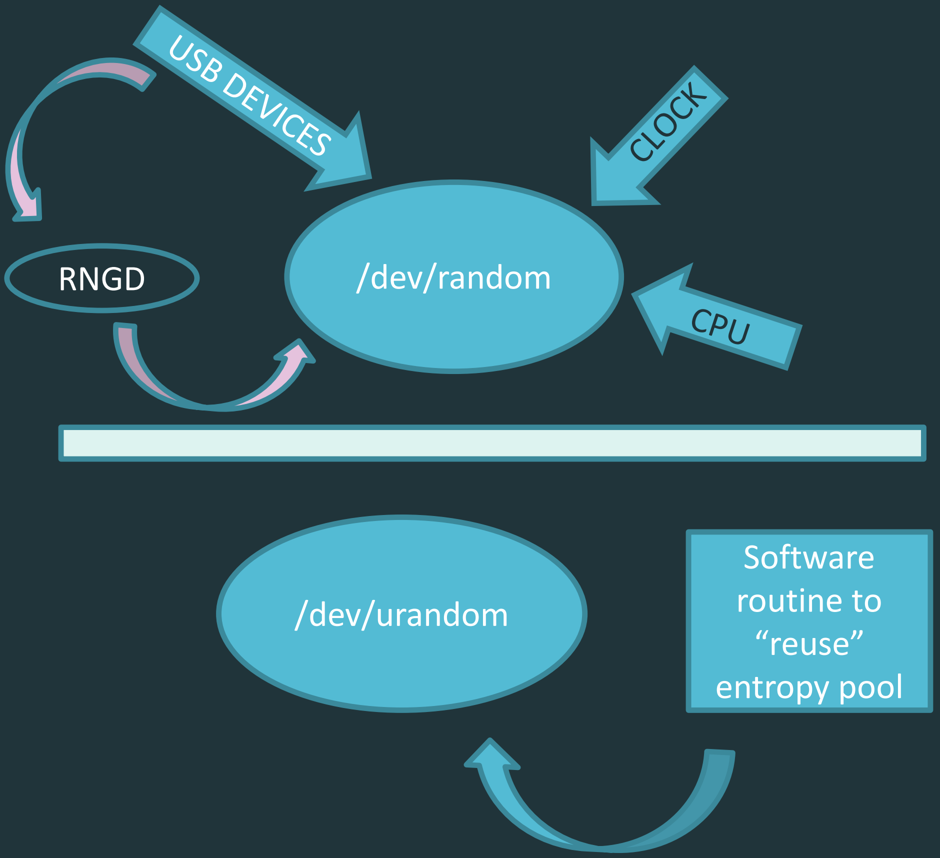
This solution acts as a "pump" for /dev/random from any available hardware device driver that can be used for refreshing the entropy on a scheduled basis.   Any "supported" hardware may be used as "input" and "output" to /dev/random.

This ensure that the entropy pool is being populated by "environmental" processes and NOT software pseudo random generators to gain a high confidence in the level of security provided by the entropy pool.

This tool is available on most UNIX/Linux Flavors (RHEL/CentOS/SuSE/Ubuntu/etc.) or can be downloaded/compiled for the UNIX/Linux OS.

## No financial cost for use or deployment of this tool.

This tool is provided with a configuration file, that allows an administrator to define how often to "prime the pump" to refresh /dev/random.   The configuration file is available under  /etc/sysconfig/rngd



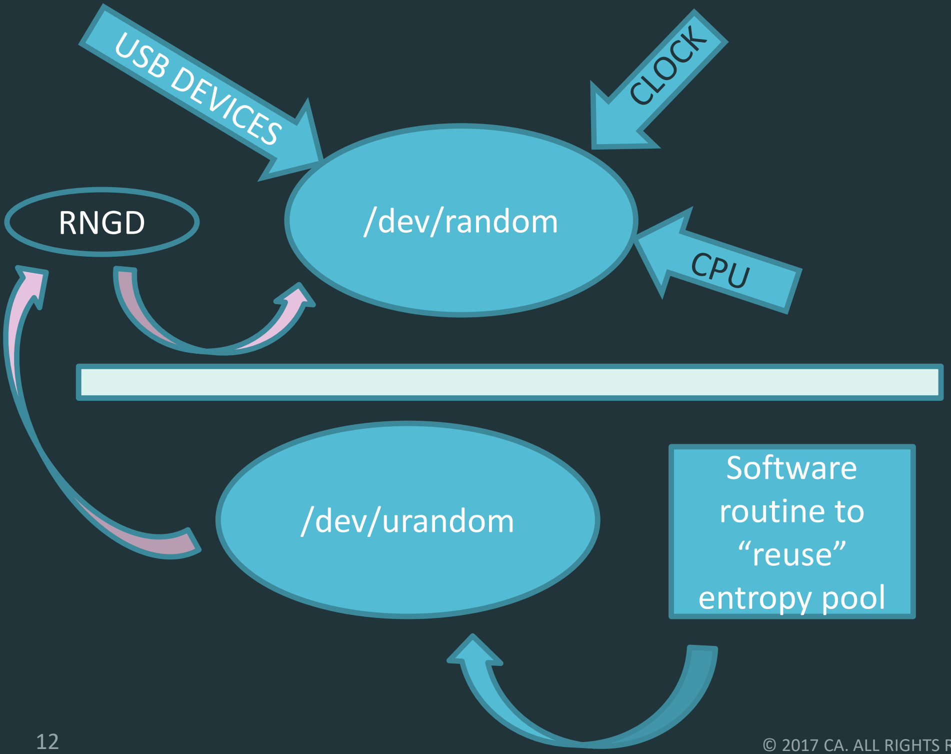

## RNGD & Entropy Pool with Hardware Device

- RNGD daemon/service will "pull" randomness from an existing "supported" physical asset and push this into the **/dev/random** entropy pool.

Settings within RNGD config file may limit that only ½ of the Entropy Pool is populated via this daemon, to avoid overwhelming/ dominating the Entropy Pool with data from one hardware device.

# RNGD & Entropy Pool with no hardware devices

1- Acceptable for DEV/QA/TEST environments
2- Acceptable for Production Non-FIPS environments after client's security/architect team has reviewed, validated against client's security policy, and performed an assessment.




- If no hardware devices are available, then RNGD may be configured to pull entropy from the pseudo device driver. This will provide an entropy pool that is partly filled (1/2) by environment and partly (1/2) by software pseudo encryption processes.

- This process is consider to be generally acceptable for non-Production systems & non-FIPS Production systems, to improve performance for solutions that use cryptography, e.g. Apache, J2EE platforms (RHEL JBOSS, IBM WebLogic, Oracle Websphere), Web Access Control solutions, SSL/TLS, LDAP(S), etc.

Example:
EXTRAOPTIONS="-i -o /dev/random -r /dev/urandom-t 10 -W 2048"

# Alternative Methods

- HAVEGED (See notes further down in deck)
- Alternatives offered by JVM Vendors
  – See Entropy and JVM page for full list
- Alternatives viewed from search results on Google consolidate down to one common "software hack"
  – Not recommended:
  – This is not the best or a good answer, as it remove ALL environmental noise, and constantly reuses the same encryption pool; and will impact FIPS compliance for ALL software.
    - mv /dev/random /dev/random.org
    - ln-s /dev/urandom/dev/random

# Check what application/ solutions are using /dev/random or /dev/urandom

- Don’t guess, find out what application(s) are using the Entropy Pool.

  – **lsof | grep-E  "/dev/[u]{0,1}random"**





ca technologies

# Entropy and JVM Notes

- BEST: (All environments) {Impact: All solutions + JVM} {FIPS compliant}

–Use hardware and the RNGD daemon to keep /dev/random populated. Existing hardware may be sufficient, so test this first.

-Use EGD daemons that harvest volatile hardware (HAVEGED) to keep /dev/random populated. {Not clear if this is FIPS compliant}

- BETTER: (DEV/TEST/QA/**PROD** environments) {Impact: JVM only} {FIPS compliant}

–Change the java configuration in a way that '/dev/urandom' is not mapping directly to '/dev/random'.

–Change the file $JAVA_HOME/jre/lib/security/java.security: securerandom.source=file:/dev/urandom into securerandom.source=file:/dev/./urandom

–{/dev/urandomdoesn't work due to unknown path issue; must use .}

- BETTER: (DEV/TEST/QA/**PROD** environments) {Impact: JVM only} {FIPS compliant}

–Add an Java option during startup of the JVM: (Oracle Recommendation): **-Djava.security.egd=file:/dev/./urandom**

•{/dev/urandomdoesn't work due to unknown path issue; must use .}

- GOOD: (DEV/TEST/QA) {Impact: All solutions + JVM} {FIPS compliant}

–Use with RNGD service until hardware can be obtained (NOTE: Using –r /dev/urandom is NOT FIPS compliant, but this give ½ environmental randomness, which is still pretty good}

•rngd -r /dev/urandom -o /dev/random -t 10

–{make the number lower to increase refresh of entropy pool}

•Edit /etc/sysconfig/rngd **EXTRAOPTIONS="-i -o /dev/random -r /dev/urandom-t 10 -W 2048"**

- OK-TESTING ONLY: (DEV/TEST/QA environments) {Impact: All solutions + JVM} {NOT FIPS compliant}

Not Recommended: This is not the best or a good answer, as it remove ALL environmental noise, and constantly reuses the same encryption pool.

{This will impact FIPS compliance for ALL software; including SSHD/LIBCRYPT functions}

•mv /dev/random /dev/random.org

•ln-s /dev/urandom/dev/random

- JUMP START Randomness pool: {All environments; add process to boot rc script of OS}

–dd if=/dev/zero of=filename.iso bs=1G count=50 {where filename.isois any large file}





ca technologies

# Example of OS RNGD Configuration File
## /etc/sysconfig/rngd

# Add extra options here
# Try first with NO extra options to see if there are any issues
#EXTRAOPTIONS="”

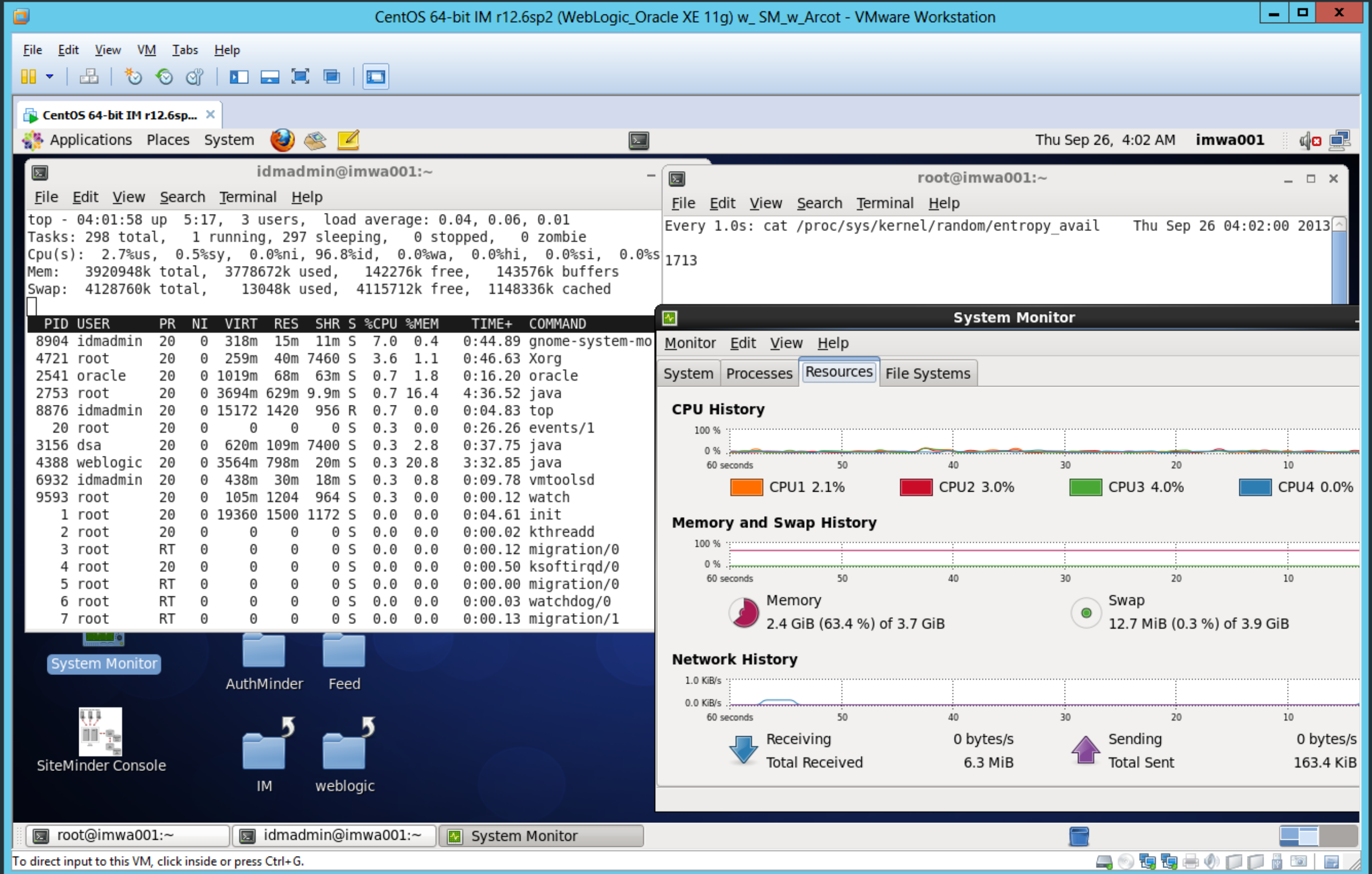
# If Hardware exist and check entropy with the following command:
#                    watch -n 1 cat /proc/sys/kernel/random/entropy_avail     shows less than 1000  &&
#                    lsof| grep  -E "/dev/[u]{0,1}random"      and this show the service under question
# then set the refresh time from default of 60 seconds to 10 second (may go lower)
#EXTRAOPTIONS="-t 10 -W 2048 »
#
# Used for WebAppServersJVM, CA IM JCS (IAMCS), CA Siteminder

EXTRAOPTIONS="-i  -r /dev/urandom  -o  /dev/random  -t 10 -W 2048"

or

EXTRAOPTIONS="-r /dev/urandom -o /dev/random -W 4096"

Example of using the following variables:
EXTRAOPTIONS="-i -o /dev/random -r /dev/urandom-t 10 -W 2048“ will give a large pool of 130-1800 to use. Very adequate






technologies

# Is /dev/urandom acceptable to use for FIPS processes?

- According to the NIST site for certification of FIPS-140-2, many vendors are using /dev/urandom as their seed for their various security modules. /dev/urandom is considered to have high security confidence if it is ONLY populated via /dev/random.
  - Red Hat Enterprise Linux 6.2 OpenSSH Server Cryptographic Module v2.1 http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1792.pdf

- FIPS-140-2 Certification requires use of a pseudo random number and the solution to inspect the "randomness" of the pseudo random number prior to consumption.

- What is NOT considered to have high security confidence:
  - Moving the OS security layer of /dev/random and replacing it with a soft link from /dev/urandom
  - Using a process to replenish /dev/random from data from /dev/urandom (that was previously fed by /dev/random – basically using old data)

# Is /dev/urandom acceptable to use for normal encryption (non-FIPS) processes?

- http://man7.org/linux/man-pages/man4/random.4.html
  - Or execute **man 4 random (2013-03-05)**

- Usage

If you are unsure about whether you should use /dev/random or /dev/urandom, then probably you want to use the latter. As a general rule, /dev/urandom should be used for everything except long-lived GPG/SSL/SSH keys.

Software that reads from the /dev/urandom device will not be blocked, waiting for more entropy.

As a result, if there is not sufficient entropy in the entropy pool, the returned values are theoretically vulnerable to a cryptographic attack on the algorithms used by the driver.

Knowledge of how to do this is not available in the current non-classified literature, but it is theoretically possible that such an attack may exist. If this is a concern in your application, use /dev/random instead.





ca technologies

# References

RFC 1750, "Randomness Recommendations for Security" (Dec. 1994)

- http://www.ietf.org/rfc/rfc1750.txt

Analysis of the Linux Random Number Generator (Mar. 2006)

- http://www.pinkas.net/PAPERS/gpr06.pdf

IBM Description of random / urandom in AIX

- http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/random.htm

HP-UX Strong Random Number Generator

- https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=KRNG11I

PRNGD - Pseudo Random Number Generator Daemon

- http://prngd.sourceforge.net/00README

DIEHARDER TOOL TO CHECK RANDOMNESS

- http://www.phy.duke.edu/~rgb/General/dieharder/dieharder.html
- http://www.phy.duke.edu/~rgb/General/dieharder.php

NIST Example of FIPS-140-2 Certification / Red Hat Enterprise Linux 6.2 OpenSSH Server Cryptographic Module v2.1 (10/2012)

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1792.pdf

HAVEGED - Newer EGD Daemon / "Harvesting Hardware Volatile States" & Algorithm

- http://www.issihosts.com/haveged/ & https://www.irisa.fr/caps/projects/hipsor/





ca technologies

# Commands / Tools to test Entropy Daemons

DIEHARDER TOOL TO CHECK RANDOMNESS of /dev/urandom

- http://www.phy.duke.edu/~rgb/General/dieharder/dieharder.html
- http://www.phy.duke.edu/~rgb/General/dieharder.php
- dieharder -d 1 -g XX -t 1000000
  - Where XX is the number for /dev/urandom
  - May download with yum install dieharder

A view in to /dev/urandom

**cat /dev/urandom | head -n 10 | sha1sum | awk '{print $1}**

```
[root@imwa001 weblogic]# cat /dev/urandom | head -n 10 | sha1sum | awk '{print $1}'
55ee44e953a39eca74641025f038e7fa86a7d7d3
[root@imwa001 weblogic]# cat /dev/urandom | head -n 10 | sha1sum | awk '{print $1}'
85f3b7397822c5e6110161acf9a214d766cc5b3d
[root@imwa001 weblogic]# cat /dev/urandom | head -n 10 | sha1sum | awk '{print $1}'
30bc813dc740c7df5886dcb68ba6d674ba198482
```

**watch -n 1 cat /proc/sys/kernel/random/entropy_avail**

```
Every 1.0s: cat /proc/sys/kernel/random/entropy_avail     Wed Oct  2 14:43:32 2013
1990
```

**lsof | grep -E "/dev/[u]{0,1}random”**

```
rngd     2034  root      3r    CHR  1,9  0t0  3847 /dev/urandom
rngd     2034  root      4u    CHR  1,8  0t0  3846 /dev/random
java     2700  root      15r   CHR  1,8  0t0  3846 /dev/random
java     2700  root      16r   CHR  1,9  0t0  3847 /dev/urandom
java     2700  root      24r   CHR  1,8  0t0  3846 /dev/random
java     3099  dsa       28r   CHR  1,8  0t0  3846 /dev/random
java     3099  dsa       29r   CHR  1,9  0t0  3847 /dev/urandom
java     3099  dsa       33r   CHR  1,9  0t0  3847 /dev/urandom
java     3099  dsa       34r   CHR  1,9  0t0  3847 /dev/urandom
java     3099  dsa       35r   CHR  1,9  0t0  3847 /dev/urandom
java     3099  dsa       54r   CHR  1,9  0t0  3847 /dev/urandom
java     3099  dsa       62r   CHR  1,9  0t0  3847 /dev/urandom
httpd    4104  root      10r   CHR  1,9  0t0  3847 /dev/urandom
java     4364  weblogic  277r  CHR  1,8  0t0  3846 /dev/random
java     4364  weblogic  278r  CHR  1,9  0t0  3847 /dev/urandom
java     4364  weblogic  299r  CHR  1,8  0t0  3846 /dev/random
java     4364  weblogic  300w  CHR  1,8  0t0  3846 /dev/random
firefox  5487  idmadmin  22r   CHR  1,9  0t0  3847 /dev/urandom
java     5889  root      252r  CHR  1,8  0t0  3846 /dev/random
java     5889  root      253r  CHR  1,9  0t0  3847 /dev/urandom
httpd    6760  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6761  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6762  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6763  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6764  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6765  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
httpd    6766  apache    10r   CHR  1,9  0t0  3847 /dev/urandom
```

# Dieharder Test of /dev/urandom

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
#    Id Test Name           | Id Test Name           | Id Test Name           #
#=============================================================================#
|   500 /dev/random         |501 /dev/urandom        |                        |
#=============================================================================#
```

```
dieharder -d 1 -g 500 -t 1000
rng_name    |rands/second|   Seed   |
    /dev/random|  1.23e+04  |1654375713|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
      diehard_operm5|   0|      1000|     100|0.67309912|  PASSED
```

```
dieharder -d 1 -g 501 -t 1000
 rng_name    |rands/second|   Seed   |
   /dev/urandom|  7.71e+04  |3445980102|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
      diehard_operm5|   0|      1000|     100|0.18839539|  PASSED
```

Condensed list to display only /dev/random and /dev/urandom

Best test is to run with 1,000,000 for validation, but this will likely take days to complete on some systems.

Test of **/dev/random** with just 1000 random test samples (instead of default 10,000 for quick check)

**PASSES randomness.**

Test of **/dev/urandom** with just 1000 random test samples (instead of default 10,000 for quick check)

**PASSES randomness**





ca technologies

# Best Solution – Tool To Harvest Volatile Hardware States for Entropy Non-FIPS requirements

WINNER

HAVEGED – Harvest Volatile HW Components EGD Daemon
http://www.issihosts.com/haveged
HAVEGED Algorithm
https://www.irisa.fr/caps/projects/hipsor/



technologies

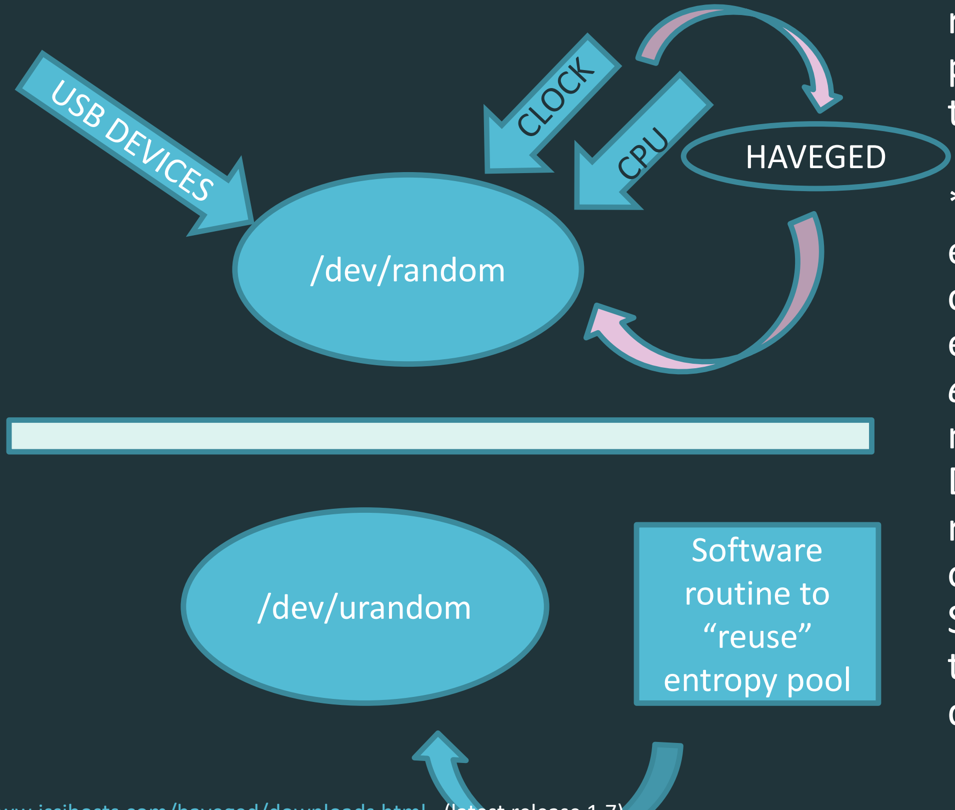
# An HW Volatile Entropy Generator: **HAVEGED**
# https://www.irisa.fr/caps/projects/hipsor/

- **HAVEGE** (**HA**rdware **V**olatile **E**ntropy **G**athering and **E**xpansion) is a user-level software unpredictable random number generator for general-purpose computers that exploits these modifications of the internal volatile hardware states as a source of uncertainty. During an initialization phase, the hardware clock cycle counter of the processor is used to gather part of this entropy.
- Its free and open source, highly recommended in the VM/Cloud community. **"GNU LESSER GENERAL PUBLIC LICENSE"** https://www.irisa.fr/caps/projects/hipsor/license.php
- It addresses the HW entropy challenge for virtualized servers.
- Two columns: A display of the random entropy pool, first when running **haveged** and then immediately after **haveged** process is killed.
- Note the decrease in entropy availability without a "pump" to /dev/random.

- #/usr/sbin/haveged -w 1024
- $ while true; do cat /proc/sys/kernel/random/entropy_avail; sleep 1; done
- 3090
- 2863
- 2626
- 2380
- 2029
- 1146
- 3963
- 3708
- 3453
- 3198
- 2965
- 2722
- 2467
- 1458
- 1203
- 4093

- # pkill -9 haveged
- $ while true; do cat /proc/sys/kernel/random/entropy_avail; sleep 1; done
- 1331
- 1076
- 821
- 575
- 334
- 131
- 150
- 179
- 132
- 178
- 186
- 133
- 160
- 131
- 165

## Non-FIPS* Compliant, acceptable for ALL environments




## Why use HAVEGED for an Entropy Pool?

- HAVEGE daemon/service will "pull" randomness from an existing "volatile" physical asset of the CPU and CLOCK; and push this into the **/dev/random** entropy pool.

* No confirmation if this is FIPS compliant, but early versions of this tool did pass randomness checks based on Lempel–Ziv compression test, entropy test, chi2 test, Monte Carlo tests from *ent*,3 the FIPS-140-2 test suite for random number generators [FIPS-140-2 2001], the DIEHARD suite and the NIST statistical suite for random number generator [p341,section 5.1 of havege-tomacs.pdf].
See links below for additional reading & a self test using dieharder toolset at the end of the deck.

http://www.issihosts.com/haveged/downloads.html (latest release 1.7)
https://www.irisa.fr/caps/projects/hipsor/publications/havege-tomacs.pdf
https://www.irisa.fr/caps/projects/hipsor/publications/havege-rr.pdf

# Comparison of HAVEGED versus OS RNGD

How does haveged run-time testing compare to rng-tools-4?
Assuming you have a rngd entropy source, the operation of rngd and haveged are similar.

The rngd test suite, FIPS140-2, is run at start-up and continuously on 20000 bit blocks; test failures on a block, discard the data and retry up to a configured limit.
Haveged runs a test suite based upon AIS31.

- AIS31 consists of a suite of tests to detected 'statistically anomalous behavior', procedure A, and a suite of more theoretical tests, procedure B.
- AIS31 procedure A consists of a disjointedness test on a block of 65k*48 bits followed by 257 repetitions of a 5 test suite run on successive 20000 bit blocks; the 5 tests consist of the 4 tests in FIPS140-1 augmented with an auto-correlation test.
- AIS32 procedure B consists of distribution tests for 10,000 runs of 1, 2, 4, 8 bit sequences followed by a 256 K bit entropy estimate (Coron's test).

Testing schedule is determined command line parameters, the defaults require both Procedure A and Procedure B to both be completed at start up and all output to pass Procedure B incrementally (data is output as long as the no individual test failures have occurred in the active test procedure operating on internally buffered data).

FIPS140-1 and FIPS140-2 differ only by acceptance limits. FIPS140-2 has slightly more stringent limits, but the FIPS140-1 limits are baked into a retry strategy that guarantees a working RNG will not shut down due to a false alarm.

- The haveged default for internal buffering ensures that no single test failure has occurred in the last ~2MB of generated data.
- When an error is detected, internal data is discarded until the active test procedure is completed; if only a single error occurred in the test procedure, the retry will initiated and output will resume only after the retry completes successfully.
- Errors that cannot be recovered by the retry procedure are fatal.

Haveged testing is performed directly on the collection buffer contents. Note that both AIS test procedures require several MB of input to complete (the procedure B requirement depends on input and is not fixed) and any test sequence including procedure B will not have any fixed buffer alignment.

http://www.issihosts.com/haveged/faq.html

# Example: Install & start haveged

```
# yum install haveged
================================================================================================
 Package              Arch                Version                Repository           Size
================================================================================================
Installing:
 haveged              x86_64              1.3-2.el6              epel                 51 k

Transaction Summary
================================================================================================
Install       1 Package(s)

Total download size: 51 k
Installed size: 159 k
Is this ok [y/N]: y
Downloading Packages:
haveged-1.3-2.el6.x86_64.rpm                                          |  51 kB     00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : haveged-1.3-2.el6.x86_64                                                   1/1
  Verifying  : haveged-1.3-2.el6.x86_64                                                   1/1

Installed:
  haveged.x86_64 0:1.3-2.el6

Complete!
```

**#/etc/init.d/haveged start**                    NOTE:   haveged starts with the following switches   -w 1024  -v 1   &  K25haveged rc script are deployed in all rc levels

**#watch -n 1 cat /proc/sys/kernel/random/entropy_avail**
1820

http://www.issihosts.com/haveged/faq.html




ca technologies

# haveged Release Notes

**Release Notes**

**Version 1.7c** Correct additional run-time test alignment problems on mips.

**Version 1.7b** Correct run-time test alignment problems on sparc and mips. Correct ppc detection in build.

**Version 1.7a** Correct VPATH issues and modify check target to support parallel builds and changes in automake 1.13 test harness. Updated sample spec file and other documentation changes.

**Version 1.7** The build and sources have been restructured to provide a devel package containing the haveged RNG implementation. Updated documentation, man(8) and man(3) pages, additional build and usage samples are provided. The potential for a rare syssegv resulting from the 1.6 changes has been removed.

**Version 1.6a** Fix typo that broke generic build procedure.

**Version 1.6** The run time test implementation has been corrected to remove an alignment fault that appeared in AIS test0 on arm64 hosts. The build procedure for clock_gettime() support has been altered to provide better control (now an override for all architectures) and correctly determine dependencies. Minor typos and inaccuracies in the source and man have been corrected.

**Version 1.5** A run time test option has been added to haveged that enables the execution of one or both of the principle AIS-31 test suites at haveged initialization and/or continuously during subsequent output. The command option permits the run time tests to be configured to trade off test overhead with test rigor to meet differing application needs. Reasonable default values are provided for daemon and direct invocations. For further details on the testing implementation see the haveged documentation. Several changes have also been made to make haveged work better with both the systemd and sysv init systems.

**Version 1.4** The haveged build has been extended to support s390 and 'generic' architectures based upon clock_gettime(). A general cleanup of the build scripts includes the ability to install non-RedHat init scripts without patching the build. The haveged collection loop has been rewritten to support multiple instances and add additional diagnostics which are being used to further improve haveged. Tuning logic for the collection has been totally rewritten to replace buggy cpuid code and incorporate additional information obtained from the /proc and /sys file systems. An experimental multi-threaded option is also provided for those hoping to spread haveged cpu load over multi-processes.

**Version 1.3** Haveged has been reorganized to allow its collection mechanism to be better accessed directly through the file system. This reorganization includes the option to suppress the daemon interface in the build so that haveged can now be used in those circumstances where the use of /dev/random is unavailable or inappropriate. This also means that haveged can now be built and used on non-linux systems. For example, the current tarball builds unmodified in mingw on Windows. A new command argument has been added to provide more precise control over file system output including unlimited piped output. The new man page provides many examples of how the new haveged file output features can be used in a linux environment.

**Version 1.2** After quite a while, I finally returned to modernizing the build. If you have a recent compiler, the build will use compiler intrinsics to replace the previous inline assembly. This is still somewhat experimental, but may help with build reliability. There are a couple of other features still in the works, but the move to intrinsic had been sitting around for a while and it was time to push it out.

http://www.issihosts.com/haveged/faq.html

# haveged man page

**haveged(8)** SYSTEM ADMINISTRATION COMMANDS **haveged(8)**

NAME haveged - Generate random numbers and feed linux random device.

SYNOPSIS haveged [options]

DESCRIPTION

The HAVEGE (HArdware Volatile Entropy Gathering and Expansion) algorithm harvests the indirect effects of hardware events on hidden processor state (caches, branch predictors, memory translation tables, etc) to generate a random sequence. The effects of interrupt service on processor state are visible from userland as timing variations in program execution speed. Using a branch-rich calculation that fills the processor instruction and data cache, a high resolution timer source such as the processor time stamp counter can generate a random sequence even on an "idle" system.

In Linux, the hardware events that are the ultimate source of any random number sequence are pooled by the /dev/random device for later distribution via the device interface. The standard mechanism of harvesting randomness for the pool may not be sufficient to meet demand, especially on those systems with high needs or limited user interaction. Haveged provides a daemon to fill /dev/random whenever the supply of random bits in /dev/random falls below the low water mark of the device.

Haveged also provides a direct file system interface to the collection mechanism that is also useful in other circumstances where access to the dev/random interface is either not available or inappropriate.

In either case, haveged uses HAVEGE to maintain a 1M pool of random bytes consumed by the interface. The principle inputs to havaged are the sizes of the processor instruction and data caches used to setup the HAVEGE collector. The haveged default is a 4kb data cache and a 16kb instruction cache. On machines with a cpuid instruction, haveged will attempt to select appropriate values from internaltables.

Although CISC architectures appear insensitive to tuning parameters, there is no guarantee that manual tuning of the algorithm may not be required under some circumstances. The output of the HAVEGE random number generator should be verified on any installation before the haveged is put into production.

OPTIONS

-d nnn, --data=nnn Set data cache size to nnn KB. Default is 16 or as determined by cpuid.

-f file, --file=file Set output file path for non-daemon use. Default is "sample", use ’-’ for stdout.

-i nnn, --inst=nnn Set instruction cache size to nnn KB. Default is 16 or as determined by cpuid.

-n nnn, --number=nnn Set number of bytes written to the outputfile. The value may be specified using one of the suffixes k, m, g, or t. The upper bound of this value is "16t" (2^44 Bytes = 16TB). A value of 0 indicates unbounded output and forces output to stdout.

-r n, --run=n Set run level for daemon interface:

n = 0 Run as daemon - must be root. Fills /dev/random when the supply of random bits falls below the low water mark of the device. This argument is required if the daemon interface is not present. If the daemon interface is present, this takes precedence over any -r value.

n = 1 Display configuration info and terminate.

n > 1 Write <n> kb of output. Deprecated (use -n instead), only provided for backward compatibility.

-v n, --verbose=n Set output level 0=minimal, 1=config/fill items, use -1 for all diagnostics.

-w nnn, --write=nnn Set write_wakeup_threshold of daemon interface to nnn bits. Applies only to run level 0.

-?, --help This summary of program options.

# Haveged test with Dieharder Test Tool

**#haveged -n 0 | dieharder -g 200 –a** **N**ote: This will redirect haveged to redirect to standard out and be piped to the dieharder tool for all tests

```
Writing unlimited bytes to stdout
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  1.26e+07  |1852405611|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.57204797|  PASSED
      diehard_operm5|   0|   1000000|     100|0.22740140|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.34454555|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.72587068|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.87414660|  PASSED
        diehard_opso|   0|   2097152|     100|0.90934123|  PASSED
        diehard_oqso|   0|   2097152|     100|0.79747155|  PASSED
         diehard_dna|   0|   2097152|     100|0.15915145|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.80153306|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.28228396|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.59549008|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.60221896|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.73160681|  PASSED
     diehard_squeeze|   0|    100000|     100|0.21997442|  PASSED
        diehard_sums|   0|       100|     100|0.33592286|  PASSED
        diehard_runs|   0|    100000|     100|0.16586980|  PASSED
        diehard_runs|   0|    100000|     100|0.19743906|  PASSED
       diehard_craps|   0|    200000|     100|0.93811739|  PASSED
       diehard_craps|   0|    200000|     100|0.77791422|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.70035080|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.88573232|  PASSED
         sts_monobit|   1|    100000|     100|0.20199896|  PASSED
            sts_runs|   2|    100000|     100|0.50203144|  PASSED
          sts_serial|   1|    100000|     100|0.57145849|  PASSED
          sts_serial|   2|    100000|     100|0.99728890|   WEAK
          sts_serial|   3|    100000|     100|0.39048937|  PASSED
          sts_serial|   3|    100000|     100|0.47442603|  PASSED
```

```
Writing unlimited bytes to stdout
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  1.26e+07  |1852405611|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
          sts_serial|   4|    100000|     100|0.53625707|  PASSED
          sts_serial|   4|    100000|     100|0.87733703|  PASSED
          sts_serial|   5|    100000|     100|0.77618814|  PASSED
          sts_serial|   5|    100000|     100|0.95254941|  PASSED
          sts_serial|   6|    100000|     100|0.90680814|  PASSED
          sts_serial|   6|    100000|     100|0.90834109|  PASSED
          sts_serial|   7|    100000|     100|0.99152198|  PASSED
          sts_serial|   7|    100000|     100|0.97731071|  PASSED
          sts_serial|   8|    100000|     100|0.99992149|   WEAK
          sts_serial|   8|    100000|     100|0.94573178|  PASSED
          sts_serial|   9|    100000|     100|0.67810844|  PASSED
          sts_serial|   9|    100000|     100|0.69787074|  PASSED
          sts_serial|  10|    100000|     100|0.47718471|  PASSED
          sts_serial|  10|    100000|     100|0.15000932|  PASSED
          sts_serial|  11|    100000|     100|0.83363869|  PASSED
          sts_serial|  11|    100000|     100|0.76524505|  PASSED
          sts_serial|  12|    100000|     100|0.77666185|  PASSED
          sts_serial|  12|    100000|     100|0.81076453|  PASSED
          sts_serial|  13|    100000|     100|0.65461525|  PASSED
          sts_serial|  13|    100000|     100|0.41736194|  PASSED
          sts_serial|  14|    100000|     100|0.76388123|  PASSED
          sts_serial|  14|    100000|     100|0.95750449|  PASSED
          sts_serial|  15|    100000|     100|0.44765617|  PASSED
          sts_serial|  15|    100000|     100|0.95942496|  PASSED
          sts_serial|  16|    100000|     100|0.96609352|  PASSED
          sts_serial|  16|    100000|     100|0.29583714|  PASSED
         rgb_bitdist|   1|    100000|     100|0.88305416|  PASSED
         rgb_bitdist|   2|    100000|     100|0.99299193|  PASSED
         rgb_bitdist|   3|    100000|     100|0.65802931|  PASSED
         rgb_bitdist|   4|    100000|     100|0.09075015|  PASSED
         rgb_bitdist|   5|    100000|     100|0.83084595|  PASSED
         rgb_bitdist|   6|    100000|     100|0.94459210|  PASSED
         rgb_bitdist|   7|    100000|     100|0.95300756|  PASSED
```

To view dieharder table -g ?
To view dieharder test -l



ca technologies

# Haveged test with Dieharder Test Tool - Cont

```
Writing unlimited bytes to stdout
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  1.26e+07  |1852405611|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
         rgb_bitdist|   8|    100000|     100|0.33944028|  PASSED
         rgb_bitdist|   9|    100000|     100|0.80409346|  PASSED
         rgb_bitdist|  10|    100000|     100|0.94020509|  PASSED
         rgb_bitdist|  11|    100000|     100|0.34379239|  PASSED
         rgb_bitdist|  12|    100000|     100|0.90682130|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.72170090|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.23678838|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.26514932|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.04750104|  PASSED
    rgb_permutations|   2|    100000|     100|0.30467809|  PASSED
    rgb_permutations|   3|    100000|     100|0.31786194|  PASSED
    rgb_permutations|   4|    100000|     100|0.85236736|  PASSED
    rgb_permutations|   5|    100000|     100|0.15300707|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.38856307|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.71141770|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.86041387|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.16291541|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.29922795|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.44457983|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.54351269|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.78409626|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.52550097|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.45489981|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.36117553|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.88815984|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.75740093|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.59917945|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.51503885|  PASSED
      rgb_lagged_sum|  15|   1000000|     100|0.12363640|  PASSED
      rgb_lagged_sum|  16|   1000000|     100|0.76457875|  PASSED
      rgb_lagged_sum|  17|   1000000|     100|0.28353930|  PASSED
      rgb_lagged_sum|  18|   1000000|     100|0.20149867|  PASSED
      rgb_lagged_sum|  19|   1000000|     100|0.74975963|  PASSED
```

```
Writing unlimited bytes to stdout
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  1.26e+07  |1852405611|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
      rgb_lagged_sum|  20|   1000000|     100|0.96746973|  PASSED
      rgb_lagged_sum|  21|   1000000|     100|0.09536836|  PASSED
      rgb_lagged_sum|  22|   1000000|     100|0.69548033|  PASSED
      rgb_lagged_sum|  23|   1000000|     100|0.54947341|  PASSED
      rgb_lagged_sum|  24|   1000000|     100|0.12359811|  PASSED
      rgb_lagged_sum|  25|   1000000|     100|0.77946198|  PASSED
      rgb_lagged_sum|  26|   1000000|     100|0.17835252|  PASSED
      rgb_lagged_sum|  27|   1000000|     100|0.05929038|  PASSED
      rgb_lagged_sum|  28|   1000000|     100|0.09040543|  PASSED
      rgb_lagged_sum|  29|   1000000|     100|0.15581580|  PASSED
      rgb_lagged_sum|  30|   1000000|     100|0.49753473|  PASSED
      rgb_lagged_sum|  31|   1000000|     100|0.36269951|  PASSED
      rgb_lagged_sum|  32|   1000000|     100|0.13122735|  PASSED
     rgb_kstest_test|   0|     10000|    1000|0.35487536|  PASSED
     dab_bytedistrib|   0|  51200000|       1|0.09280526|  PASSED
             dab_dct| 256|     50000|       1|0.93192984|  PASSED
Preparing to run test 207.  ntuple = 0
        dab_filltree|  32|  15000000|       1|0.23519794|  PASSED
        dab_filltree|  32|  15000000|       1|0.06541734|  PASSED
Preparing to run test 208.  ntuple = 0
       dab_filltree2|   0|   5000000|       1|0.46434559|  PASSED
       dab_filltree2|   1|   5000000|       1|0.15917186|  PASSED
Preparing to run test 209.  ntuple = 0
        dab_monobit2|  12|  65000000|       1|0.83659955|  PASSED
```

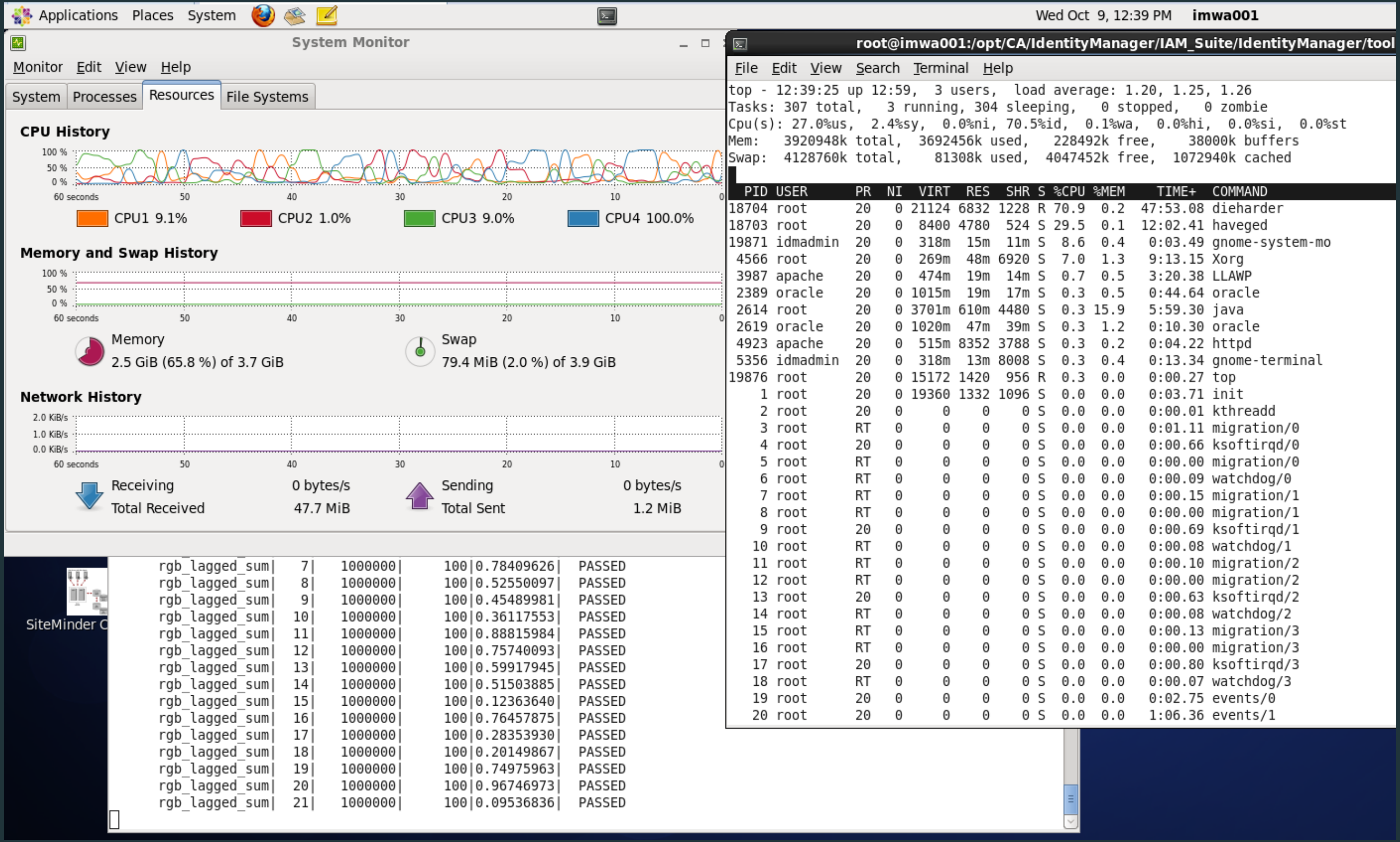
“These tests for randomness are not a proof of randomness. However, it may be considered as an indicator that finding and exploiting a bias in the generated sequences would be very difficult, particularly for a nondeterministic random number generator.”

Ref: https://www.irisa.fr/caps/projects/hipsor/publications/havege-tomacs.pdf




ca technologies

# CENTOS 6.4 x64 dieharder test of haveged daemon



# Questions?