Kernel Address Space Layout Randomization (kASLR) Test Design Document (CGE7)

Prem Karat (pkarat@mvista.com)

This document has following contents

I. Test Design

II. Sample test results

I. Test Design

The test suite has three different tests,

1. test-sec-aslr-kernel.py,
2. test-sec-aslr-klm.py,
3. test-sec-aslr-dist.py.

They have to be executed independently.

Test-SEC-ASLR-KERNEL

The objective is to test if kernel text, data, bss region is randomized each time the kernel boots. This means, the symbols exported by kernel in /proc/kallsyms should display unique address each time the kernel reboots.

NOTE: By default in MV CGE7 Kernel the CONFIG_KALLSYMS_ALL is disabled. To see all the symbols (data and BSS) ensure this kernel config is enabled for testing purpose. This option depends on CONFIG_EXPERT to be enabled as well.

An exported kernel text symbol will look like

ffffffff81000000 ***T*** *startup_64*

ffffffff81003000 ***t*** *native_read_cr4*

which is of the format,

virtual_address symbol_type symbol_name

Similarly a data symbol will look like

000000000000c4c0 ***D*** *irq_stack_ptr*

000000000000db80 ***d*** *ipi_mask*

BSS symbols will look like

ffffffff81dd4000 ***B*** *__bss_start*

ffffffff81dd7000 ***b*** *bm_pte*

This test unit has 3 tests

a. Test kernel text region randomization

b. Test kernel data region randomization

c. Test kernel bss region randomization

NOTE: Since this test for all the exported text, data and bss symbols (in kallsyms), this would include the symbols exported from ***all the modules*** inserted in the kernel at the time of running the test.

The test execution flow will look like this.

a. Reboot Kernel 'n' times

b. For every reboot, save /proc/kallsyms in a text file (save the file with unique name, like kallsyms1.txt, kallsyms2.txt etc..)

c. Now copy all these text files to a defined location in test suite and execute the test-sec-aslr-kernel.py.

The test-sec-aslr-kernel.py has 3 tests as listed above. For each symbol, it will test if the virtual address is random or static. The test would **PASS** if the address are random across all reboots for all text, data and bss symbols.

A sample test result is as follows

test-sec-alsr-kernel.py:

Adress Space Randomization	Result
Kernel Text	PASS
Kernel Data	PASS
Kernel BSS	FAIL

Detailed logs available in logs/test-sec-alsr-kernel.log .

There is also a verbose option available through which you could get the following statistics

Total Kernel <Text|Data|BSS> Symbols

Kernel <Text|Data|BSS> Symbols with unique address

Kernel <Text|Data|BSS> Symbols with Duplicate address

Test-SEC-ASLR-KLM

The objective here is to test if the symbols exported by the kernel module displays unique address each time the module is inserted after system boot. Module text, data and bss region is tested for randomization.

For testing purpose, we have developed a kernel module that exports 3 types of symbols, kernel text (A kernel function), kernel data (Initialized global variable), and kernel BSS symbol (Uninitialized global variable).

This test unit again tests for module text, data and bss region randomization. The test execution flow is as follows

a. Reboot kernel 'n' times

b. For every reboot, insert the test kernel module and after the module is inserted, save /proc/kallsyms in a text file (save the file with unique name, like klm1.txt, klm2.txt etc..)

c. Now copy all these text files to a defined location in test suite and execute the test-sec-aslr-klm.py

The test would test if the virtual address of each of these exported symbols (text, data and bss) from the module are random across every module insertion after kernel reboot.

NOTE: Its important that the module is inserted after every kernel reboot. We have observed that if the module is inserted, then removed and then inserted back again (without rebooting kernel), then the virtual address of the exported symbols from the module are random irrespective of whether kASL is enabled or disabled. However if kASLR is disabled and if you reboot the kernel and insert the module first time after every reboot, then we have found that the virtual address remain static across reboots.

A sample test result is as follows

test-sec-alsr-kernel.py:

Address Space Randomization	Result
KLM Text	PASS
KLM Data	PASS
KLM BSS	FAIL

Detailed logs in logs/test-sec-alsr-klm.log

Test-SEC-ASLR-DIST

The objective here is to test if the binaries in FOSS packages is enabled for randomization (PIC/PIE). We have designed 4 tests here.

1. Test if we can build a default-image (bitbake) with '-fPIC -pie' compiled for all the packages in default image. This includes FOSS + additional packages. This is achieved by using the following variable in bitbake local.conf

TARGET_CC_ARCH += "-fPIC -pie"

This would ensure that all the FOSS packages + any additional packages included in default-image or cge-complete-image is build for randomization.

2. This test will test if the binaries & libraries of each FOSS package are indeed compiled for randomization by reading the ELF header of each binary & library. The ELF header of the binary & library should read the type as DYN and not EXEC.

Type: DYN (PASS)

Type: EXEC (FAIL)

If TYPE: EXEC then the binary/library is static and not enabled for randomization.

3. **NOTE:** This is an additional test. This test would test for all the binaries & libraries for randomization under

['/bin', '/sbin', '/usr/bin', '/usr/sbin', '/lib',

'/lib64', '/lib32', '/usr/lib',

'/usr/lib32', '/usr/lib64']

This would ensure that binaries or libraries in any packages build outside of the FOSS list or build locally will be tested for randomization. This above python list is configurable and can be extended to include additional directories like '/usr/local/bin' and '/opt' etc.,

4. This test will start each process under 'cmds' in config.py and check for the text address region of each process under /proc/<pid>/maps. This test will execute each process 3 times and check if unique address are allocated each time.

This list can be expanded by appending the 'cmds' list variable in config.py

```
cmds = [
    ['netstat'],
    ['tcpdump'],
    ['virsh'],
    ['portmap', '-f'],
    ['ping', 'localhost'],
    ['strace', '-p', '1'],
    ['ltrace', '-p', '1'],
    ['ssh', 'root@localhost'],
    ['tracepath', '172.29.251.3'],
    ['traceroute', '172.29.251.3']
]
```

Note: Each cmd is given as a list, with each parameter as list variable. Also since we don't know the heap, stack, mmap/vdso, libs region for each of these binaries above, we are not testing these regions. However the assumption here is, since we are extensively testing ASLR (user space randomization) feature in our distro by using a userspace program that will make use of a stack, heap, mmap, libs, vdso & brk regions, we are relying that if the text region of these binaries are randomized then other regions too will be randomized.

Sample test results:

Test 2:

PACKAGE	PIC/PIE Enabled
ssh	PASS
telnet-bsd	FAIL
.....	....
.....	....

Total Packages: 70

Packages enabled for PIC/PIE: 15

Packages disabled for PIC/PIE: 55

Test 3:

BINARY/LIBS	PIC/PIE Enabled
paxcpio	PASS
rnano	FAIL
.....	.....
.....	.....

Total binaries, shared libs scanned: 18382

Binaries, libs enabled for PIC/PIE: 5706

Binaries, libs disabled for PIC/PIE: 12676

Test 4:

PROCESS	Result	Iteration 1	Iteration 2
ssh	PASS	fea8-fea4	44b4-44b4
netstat	FAIL	0000-1a00	0000-1a00
virsh	FAIL	0000-4500	0000-4500
.....	.....	.....	.....
.....	.....	.....	.....