NAME

aslr - Address Space Layout Randomization

SYNOPSIS

#include <sys/types.h>
#include <sys/pax.h>

In the kernel configuration file:
options PAX
options PAX_ASLR

DESCRIPTION

Introduction

Security in FreeBSD is based primarily in policy-based technologies. Existing tools such as jail(4), capsicum(4), VNET(9), and the mac(4) framework can make FreeBSD-based systems quite resilient against attacks. FreeBSD lacks basic low-level exploit mitigation, such as Address Space Layout Randomization (ASLR). ASLR Randomizes the address space layout of an application, making exploitation difficult for an attacker. This manual page and the associated implementation aim to provide a secure, robust, extensible, and easily-managed form of ASLR fit for production use in FreeBSD.

General Overview

When compiled with the **option PAX_ASLR** , systems will have ASLR enabled. For systems with that kernel option enabled, if a user wants to disable ASLR for a given application, the user must force that application to opt-out. HardenedBSD has a special application called secadm for opting applications in to or out of exploit mitigation features such as ASLR.

Another kernel option, **PAX_SYSCTLS**, exposes additional sysctl(8) tunables, allowing ASLR behavior control without requiring a reboot. By default, the tunable hardening.pax.aslr.status can only be changed at boot time via /boot/loader.conf. Enabling the PAX_SYSCTLS kernel option allows a root user to modify hardening.pax.aslr.status at run time. See Appendix A for a list of all the tunables.

ASLR tunables are per-jail and each jail inherits its parent jail’s settings. Having per-jail tunables allows more flexibility in shared-hosting environments. This structure also allows a user to selectively disable ASLR for applications that misbehave. ASLR-disabled applications will still have policy-based security applied to it by virtue of being jailed.

Implementation Details

A new sysinit subroutine ID, SI_SUB_PAX, initializes ASLR system variables. Upon system boot, tunables from /boot/loader.conf are checked for validity. Any invalid values generate a warning

message to the console and the tunable is set to a sensible default.

For the sake of performance, the ASLR system relies on per-process deltas rather than calling arc4random(3) for each mapping. When a process calls execve(2), the ASLR deltas are initialized. Deltas are randomly generated for the execution base, mmap(2), and stack addresses. Only the execution base of applications compiled as Position Independent Executables (PIEs) is randomized. The execution base of non-PIE applications is not modified. The mappings of shared objects are randomized for both PIE and non-PIE applications.

The deltas are used as a hint to the Virtual Memory (VM) system. The VM system may modify the hint to make a better fit for superpages and other alignment constraints.

The delta applied to the PIE execbase is different than the delta applied to the base address of shared objects. In the Executable and Linkable File (ELF) image handler, the execution base of PIE applications is randomized by adding the delta controlled by the hardening.pax.aslr.exec_len tunable to et_dyn_addr, which is initialized to be ET_DYN_LOAD_ADDR (an architecture- dependent macro). The base address of shared objects loaded by the dynamic linker are randomized by applying the delta controlled by the hardening.pax.aslr.mmap_len tunable in **sys_mmap**(). Stack randomization is implemented using a small stack gap and a bigger random, which applies to the stack's mapping. On executable image activation, the stack delta is computed and subtracted from the top of the stack.

Shared object load order in the rtld(1) is randomized with respect to the Depenency Acyclical Graph (DAG). Randomizing the DAG helps prevent successful ROP gadget creation by further removing deterministic loading of shared objects. Often, when ROP gadgets are generated, the generation requires shared objects providing gadgets to be in the same spot in memory relative to the ASLR deltas.

APPENDIX A

NOTE: All tunables can only be changed during boot-time via */boot/loader.conf* unless the kernel has been compiled with **PAX_SYSCTLS**.

- hardening.pax.aslr.status
  - Type: integer
  - Description: Toggle system-wide ASLR protection.
  - Values:
    0 - ASLR disabled system-wide. Individual applications may *NOT* opt in.
    1 - ASLR disabled but applications may opt in.
    2 - ASLR enabled and applications may opt out.
    3 - ASLR enabled for all applications. Applications may not opt out.
  - Default: 2

- hardening.pax.aslr.exec_len
  - Type: integer
  - Description: Set the number of bits to be randomized for the PIE execbase.
  - Values:
    For 32-bit systems, minimum of 8, maximum of 21. For 64-bit systems, minimum of 16, maximum of 42.
  - Default: For 32-bit systems: 14. For 64-bit systems: 30.

- hardening.pax.aslr.mmap_len
  - Type: integer
  - Description: Set the number of bits to be randomized for mmap(2) calls.
  - Values:
    For 32-bit systems, minimum of 8, maximum of 21. For 64-bit systems, minimum of 16, maximum of 42.
  - Default: For 32-bit systems: 14. For 64-bit systems: 30.

- hardening.pax.aslr.stack_len
  - Type: integer
  - Description: Set the number of bits to be randomized for the stack.
  - Values:
    For 32-bit systems, minimum of 8, maximum of 21. For 64-bit systems, minimum of 16, maximum of 42.
  - Default: For 32-bit systems: 14. For 64-bit systems: 42.

- hardening.pax.aslr.vdso_len
  - Type: integer
  - Description: Set the number of bits to be randomized for the shared page (vdso).
  - Values:
    For 32-bit systems, minimum of 8, maximum of 21. For 64-bit systems, minimum of 16, maximum of 38.
  - Default: For 32-bit systems: 10. For 64-bit systems: 28.

- hardening.pax.aslr.map32bit_len
  - Type: integer
  - Description: Set the number of bits to be randomized for the mmap(2) calls when MAP_32BIT flag set.
  - Values:
    For 32-bit systems N/A. For 64-bit systems, minimum of 8, maximum of 27.
  - Default: For 32-bit systems: N/A. For 64-bit systems: 18.

SEE ALSO

rtld(1), mmap(2), elf(3), mac(4)

PaX ASLR, http://pax.grsecurity.net/docs/aslr.txt.

HardenedBSD, http://hardenedbsd.org/.

secadm, https://github.com/HardenedBSD/secadm.

HISTORY

On 14 May 2013, Oliver Pinter published to GitHub an initial patch to implement ASLR. His work was inspired by Elad Efrat's work in NetBSD. The patch was submitted to FreeBSD as a bug report on 24 Aug 2013. Independenty of Oliver's work, Shawn Webb posted to his tech blog that he was interested in implementing ASLR for FreeBSD. Oliver found his post and suggested that he and Shawn work together. On June 08, 2014, preparatory work was committed to HardenedBSD, adding Position-Independent Executable (PIE) support in base. PIE support was removed sometime later. On 07 Apr 2014, SoldierX agreed to sponsor the project and donated a sparc64 machine and a BeagleBone Black to Shawn Webb. This hardware was used for testing and debugging ASLR. ASLR for FreeBSD was first introduced in HardenedBSD. On 19 December 2015, HardenedBSD officially decided to halt the upstreaming process.

AUTHORS

This manual page was written by Shawn Webb. The ASLR implementation was written by Oliver Pinter and Shawn Webb.

BUGS

The original PaX implementation, from which the HardenedBSD implementation also drew inspiration, uses a special ELF process header which requires modification of executable files. The authors of the HardenedBSD implementation have deliberately chosen to go a different route based on the mac(4) framework. Support for filesystem extended attributes will be added at a later time.

The shared object load order randomization can lend itself to interesting behavior. If multiple libraries contain symbols of the same name, randomizing the order in which shared libraries get loaded can cause symbol lookups to resolve to the wrong symbol. Though incorrect resolution is rare, it is known to happen. Privoxy is one such example. Privoxy's list of dependencies are small, but two of them implement symbols of the same name. If Privoxy's dependencies are loaded in the wrong order, Privoxy will reference the wrong symbol and will crash. Shared library load order randomization can be disabled on a per-application basis with secadm for such cases. Work is underway to make shared object load order randomization more robust and prevent symbol resolution conflicts.