NAME

mitigations - FreeBSD Security Vulnerability Mitigations

SYNOPSIS

In FreeBSD, various security mitigations are employed to limit the impact of vulnerabilities and protect the system from malicious attacks. Some of these mitigations have run-time controls to enable them on a global or per-process basis, some are optionally enabled or disabled at compile time, and some are inherent to the implementation and have no controls.

The following vulnerability mitigations are covered in this document:

- Address Space Layout Randomization (ASLR)
- Position Independent Executable (PIE)
- Write XOR Execute page protection policy
- PROT_MAX
- Relocation Read-Only (RELRO)
- Bind Now
- Hardware Vulnerability Mitigation Controls
- Capsicum

Please note that the effectiveness and availability of these mitigations may vary depending on the FreeBSD version and system configuration.

DESCRIPTION

Security vulnerability mitigations are techniques employed in FreeBSD to limit the potential impact of security vulnerabilities in software and hardware. It is essential to understand that mitigations do not directly address the underlying security issues. They are not a substitute for secure coding practices. Mitigations serve as an additional layer of defense, helping to reduce the likelihood of a successful exploitation of vulnerabilities by making it more difficult for attackers to achieve their objectives.

This manual page describes the security mitigations implemented in FreeBSD to enhance the overall security of the operating system. Each mitigation is designed to protect against specific types of attacks and vulnerabilities.

SOFTWARE VULNERABILITY MITIGATIONS

Address Space Layout Randomization (ASLR)

Address Space Layout Randomization (ASLR) is a security mitigation technique that works by randomizing the memory addresses where system and application code, data, and libraries are loaded, making it more challenging for attackers to predict the memory layout and exploit vulnerabilities.

ASLR introduces randomness into the memory layout during process execution, reducing the

predictability of memory addresses. ASLR is intended to make exploitation more difficult in the event that an attacker discovers a software vulnerability, such as a buffer overflow.

ASLR can be enabled on both a global and per-process basis. Global control is provided by a separate set of sysctl(8) knobs for 32- and 64-bit processes. It can be or disabled on a per-process basis via proccontrol(1). Note that an ASLR mode change takes effect upon address space change, i.e., upon execve(2).

Global controls for 32-bit processes:

kern.elf32.aslr.enable   Enable ASLR for 32-bit ELF binaries, other than Position Independent Exectutable (PIE) binaries.

kern.elf32.aslr.pie_enable Enable ASLR for 32-bit Position Independent Executable (PIE) ELF binaries.

kern.elf32.aslr.honor_sbrk

　　　　　　　　　　　　Reserve the legacy sbrk(2) region for compatibility with older binaries.

kern.elf32.aslr.stack   If ASLR is enabled for a process, also randomize the stack location.

Global controls for 64-bit processes:

kern.elf64.aslr.enable   Enable ASLR for 64-bit ELF binaries, other than Position Independent Exectutable (PIE) binaries.

kern.elf64.aslr.pie_enable Enable ASLR for 64-bit Position Independent Executable (PIE) ELF binaries.

kern.elf64.aslr.honor_sbrk

　　　　　　　　　　　　Reserve the legacy sbrk(2) region for compatibility with older binaries.

kern.elf64.aslr.stack   If ASLR is enabled for a process, also randomize the stack location.

To execute a command with ASLR enabled or disabled:

proccontrol **-m** *aslr* [**-s** *enable | disable*] *command*

Position Independent Executable (PIE)
PIE binaries are executable files that do not have a fixed load address. They can be loaded at an arbitrary memory address by the rtld run-time linker. With ASLR they are loaded at a random address on each execution.

Write XOR Execute page protection policy
Write XOR Execute (W^X) is a vulnerability mitigation strategy that strengthens the security of the system by controlling memory access permissions.

Under the W^X mitigation, memory pages may be writable (W) or executable (E), but not both at the same time. This means that code execution is prevented in areas of memory that are designated as writable, and writing or modification of memory is restricted in areas marked for execution. Applications that perform Just In Time (JIT) compilation need to be adapted to be compatible with W^X.

There are separate sysctl(8) knobs to control W^X policy enforcement for 32- and 64-bit processes. The W^X policy is enabled by setting the appropriate allow_wx sysctl to 0.

kern.elf32.allow_wx Allow 32-bit processes to map pages simultaneously writable and executable.

kern.elf64.allow_wx Allow 64-bit processes to map pages simultaneously writable and executable.

PROT_MAX
PROT_MAX is a FreeBSD-specific extension to mmap(2). PROT_MAX provides the ability to set the maximum protection of a region allocated by mmap and later altered by mprotect. For example, memory allocated originally with an mmap prot argument of PROT_MAX(PROT_READ | PROT_WRITE) | PROT_READ may be made writable by a future mprotect call, but may not be made executable.

Relocation Read-Only (RELRO)
Relocation Read-Only (RELRO) is a mitigation tool that makes certain portions of a program's address space that contain ELF metadata read-only, after relocation processing by rtld(1).

When enabled in isolation the RELRO option provides *partial RELRO* support. In this case the Procedure Linkage Table (PLT)-related part of the Global Offset Table (GOT) (in the section typically named .got.plt) remains writable.

RELRO is enabled by default. The src.conf(5) build-time option *WITHOUT_RELRO* may be used to disable it.

BIND_NOW
The *WITH_BIND_NOW* src.conf(5) build-time option causes binaries to be built with the DF_BIND_NOW flag set. The run-time loader rtld(1) will then perform all relocation processing when the process starts, instead of on demand (on the first access to each symbol).

When enabled in combination with RELRO (which is enabled by default) this provides *full RELRO*. The entire GOT (.got and .got.plt) are made read-only at program startup, preventing attacks on the relocation table. Note that this results in a nonstandard Application Binary Interface (ABI), and it is possible that some applications may not function correctly.

Hardware vulnerability controls

See security(7) for more information.

Capsicum

Capsicum is a lightweight OS capability and sandbox framework. See capsicum(4) for more information.

HARDWARE VULNERABILITY MITIGATIONS

Recent years have seen an unending stream of new hardware vulnerabilities, notably CPU ones generally caused by detectable microarchitectural side-effects of speculative execution which leak private data from some other thread or process or sometimes even internal CPU state that is normally inaccessible. Hardware vendors usually address these vulnerabilities as they are discovered by releasing microcode updates, which may then be bundled into platform firmware updates (historically called BIOS updates for PCs).

The best defense overall against hardware vulnerabilities is to timely apply these updates when available and to disable the affected hardware's problematic functionalities when possible (e.g., CPU Simultaneous Multi-Threading). Software mitigations are only partial substitutes for these, but they can be helpful on out-of-support hardware or as complements for just-discovered vulnerabilities not yet addressed by vendors. Some software mitigations depend on hardware capabilities provided by a microcode update.

FreeBSD's usual policy is to apply by default all OS-level mitigations that do not require recompilation, except those the particular hardware it is running on is known not to be vulnerable to (which sometimes requires firmware updates), or those that are extremely detrimental to performance in proportion to the protection they actually provide. OS-level mitigations generally can have noticeable performance impacts on specific workloads. If your threat model allows it, you may want to try disabling some of them in order to possibly get better performance. Conversely, minimizing the risks may require you to explicitly enable the most expensive ones. The description of each vulnerability/mitigation indicates whether it is enabled or disabled by default and under which conditions. It also lists the knobs to tweak to force a particular status.

Zenbleed

The "Zenbleed" vulnerability exclusively affects AMD processors based on the Zen2 microarchitecture. In contrast with, e.g., Meltdown and the different variants of Spectre, which leak data by leaving

microarchitectural traces, Zenbleed is a genuine hardware bug affecting the CPU's architectural state. With particular sequences of instructions whose last ones are mispredicted by speculative execution, it is possible to make appear in an XMM register data previously put in some XMM register by some preceding or concurrent task executing on the same physical core (disabling Simultaneous Muti-Threading (SMT) is thus not a sufficient protection).

According to the vulnerability's discoverer, all Zen2-based processors are affected (see **https://lock.cmpxchg8b.com/zenbleed.html**). As of August 2023, AMD has not publicly listed any corresponding errata but has issued a security bulletin (AMD-SB-7008) entitled "Cross-Process Information Leak" indicating that platform firmware fixing the vulnerability will be distributed to manufacturers no sooner than the end of 2023, except for Rome processors for which it is already available. No standalone CPU microcodes have been announced so far. The only readily-applicable fix mentioned by the discoverer is to set a bit of an undocumented MSR, which reportedly completely stops XMM register leaks.

FreeBSD currently sets this bit by default on all Zen2 processors. In the future, it might set it by default only on those Zen2 processors whose microcode has not been updated to revisions fixing the vulnerability, once such microcode updates have been actually released and community-tested. To this mitigation are associated the following knobs:

machdep.mitigations.zenbleed.enable
>A read-write integer tunable and sysctl indicating whether the mitigation should be forcibly disabled (0), enabled (1) or if it is left to FreeBSD to selectively apply it (2). Any other integer value is silently converted to and treated as value 2. Note that this setting is silently ignored when running on non-Zen2 processors to ease applying a common configuration to heterogeneous machines.

machdep.mitigations.zenbleed.state
>A read-only string indicating the current mitigation state. It can be either "Not applicable", if the processor is not Zen2-based, "Mitigation enabled" or "Mitigation disabled". This state is automatically updated each time the sysctl *machdep.mitigations.zenbleed.enable* is written to. Note that it can become inaccurate if the chicken bit is set or cleared directly via cpuctl(4) (which includes the cpucontrol(8) utility).

The performance impact and threat models related to these mitigations should be considered when configuring and deploying them in a FreeBSD system.

SEE ALSO
>elfctl(1), proccontrol(1), rtld(1), mmap(2), src.conf(5), sysctl.conf(5), security(7), cpucontrol(8), sysctl(8)