eBPF in kernel lockdown mode

Arnaldo Carvalho de Melo <acme@redhat.com>

LPC 2020 Networking + BPF miniconf, The Internet

Abstract

Linux has a new 'lockdown' security mode where changes to the running kernel requires verification with a cryptographic signature and restrictions to accesses to kernel memory that may leak to userspace.

Lockdown's 'integrity' mode requires just the signature, while in 'confidentiality' mode in addition to requiring a signature the system can't leak confidential information to userspace.

Work needs to be done to add cryptographic signatures for eBPF bytecode. The signature is then passed to the kernel via sys_bpf() reusing the kernel module signing infrastructure.

The main eBPF loader, libbpf, may perform relocations on the received bytecode for things like CO-RE (Compile Once, Run Everywhere), thus tampering with the signature made with the original bytecode.

It is thus needed to move such modifications to the signed bytecode from libbpf to the kernel, so that it may be done after the signature is verified.

This presentation is intended to provide a problem statement, some ideas being discussed, provide a reading list, and to foster awareness about this security feature so that BPF can be used in environments where 'lockdown' mode is required.

eBPF in kernel lockdown mode

Introduction

Linux now has a new LSM (Linux Security Module) that implements restrictions on what root can do to reduce the possibility that unauthorized unsigned code runs. This module is called 'lockdown'[1].

Lockdown has two modes: the 'integrity' one that requires that the kernel, modules and whatever runs in kernel space be signed, and the 'confidentiality' mode, that in addition to requiring a signature, disables kernel features that may leak confidential information from the kernel.

This presentation is intended to provide a problem statement, some ideas being discussed, provide a reading list, and to foster awareness about this security feature so that BPF can be used in environments where 'lockdown' mode is required.

Restricting confidential information leaking

The kernel eBPF subsystem was already modified to avoid reading kernel memory[2], by restricting the bpf_probe_read() BPF helper when the kernel is in lockdown confidentiality mode.

As with other subsystems[3], maybe there are more places where such checks needs to be done, such as uprobes, kprobes, etc.

Allowing eBPF to probe in selected areas needs to be investigated so that the value of BPF tracing programs in lockdown mode can be made available. For instance, XDP programs can look at kernel memory, but in a very restricted fashion, being restricted to the network packet being processed when the XDP program runs.

Approaches helping with allowing to rule out confidential areas for eBPF programs in confidentiality mode include to "add support for privileged applications with an appropriate signature that implement policy on the userland side." [8]

Signing eBPF

The next area to look at to make eBPF usable in lockdown mode is in signature verification[4] by the kernel when loading BPF bytecode.

Initially this will be for tools that come with pre-compiled byte code[5]. Next will be tools that dynamically generate code, such as bpftrace, as these will be more difficult to address. The main eBPF loader, libbpf, may perform changes, such as relocations on the received bytecode for things like CO-RE (Compile Once, Run Everywhere)[6], which tamper with the signature made with the original bytecode.

These modifications will need to be moved from libbpf to the signed bytecode in the kernel, so that they can be performed after the signature is verified.

Here we should probably reuse the infrastructure for kernel module signing and verification. sys_bpf() can be changed to receive the signature, to check it and then to pass it to a component that will do the changes now performed in libbpf.

Moving libbpf to a User Mode Helper

Another idea that is being considered is to use a UMH (user mode helper) like with bpfilter[7] for doing the bytecode modifications. The parts of libbpf that perform the changes can be moved to this new component.

A signed userspace component would act as the helper that would do the libbpf relocations. It would receive commands thru pipes setup by the kernel after it performs signature verification.

This could happen both when sys_bpf receives any bytecode or metadata (BTF) from userspace or when treating BPF ELF files as executable files going through the kernel loader. This would help cover the whole ELF file and cover the metadata, such as , where to insert the BPF bytecode in each of the ELF sections.

Accepting unsigned eBPF programs

Further restricting what an unsigned eBPF program can do in kernel space when in lockdown mode may be an alternative mode for requiring a signature. Programs that need to use more capabilities would need to be signed. This could help with a subset of observability tools.

Annotating acceptable lockdown modes

Another feature that may be worth having would be for the tool writer to state what is permissible for a given signed BPF bytecode to do when in each of the lockdown modes. This is similar to capability dropping, where the BPF verifier would be informed on what is acceptable in each mode via some signed metadata.

Dynamic BPF programs

The dynamic case, where tools such as bpftrace are involved would come later, after the pre-built, signed bytecode problem is solved. This could involve a variation of an approach described in a recent blog post by Matthew Garrett[8]:

"Add support for privileged applications with an appropriate signature that implement policy on the userland side. This is actually possible already, though not straightforward. Lockdown is implemented in the LSM layer, which means the policy can be imposed using any other existing LSM. As an example, we could use SELinux to impose the confidentiality restrictions on most processes but permit processes with a specific SELinux context to use them, and then use EVM[9] to ensure that any process running in that context has a legitimate signature. This is quite a few hoops for a general purpose distribution to jump through."

Signing with bpftool

The bpftool utility will need to get functionality now found in the linux kernel module signature utility scripts/sign-file.c. And a companion ELF section will need to be added that contains the signature for each of the BPF ELF bytecodes and BTF sections.

We'd have a new subcommand:

```
bpftool sign file FILENAME
```

Then when libbpf loads the bytecode, it will add that signature to the bpf_attr struct passed to sys_bpf(BPF_PROG_LOAD).

Using dm-verity for integrity

Matteo Croce proposed[10] passing an fd for bytecode that is stored in a dm-verity[11] volume. This isn’t an option with changes made by libbpf prior to sending the bytecode to the kernel. But it may be a valid usecase if the relocation is done in the kernel, since the usecase seems related to what is in the “Integrity Policy Enforcement LSM (IPE)”[12] patch series.

This would be an alternative to an explicit signature for the BPF bytecode, relying on dm-verity for satisfying the integrity requirements.

Restricting new BPF helpers

In the future new BPF helpers would initially be disabled for 'confidentiality' mode. This conservative approach allows for the needed time to assess the safety of each new helper.

References:

[1] "Why lock down the kernel?", Matthew Garrett

https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Why-lock-down-the-kernel_Matthew-Garrett.pdf

[2] "bpf: Restrict bpf when kernel lockdown is in confidentiality mode", David Howells
https://git.kernel.org/torvalds/c/9d1f8be5cf42

[3] "powerpc/xmon: Restrict when kernel is locked down", Christopher M. Riedl
https://git.kernel.org/torvalds/c/69393cb03ccd
"ACPI: configfs: Disallow loading ACPI tables when locked down", Jason A. Donenfeld
https://git.kernel.org/torvalds/c/75b0cea7bf30

[4] "kexec: do not verify the signature without the lockdown or mandatory signature"
http://git.kernel.org/torvalds/c/fd7af71be542

[5] "BPF: The Status of BTF", description of runqslower, Arnaldo Carvalho de Melo

http://vger.kernel.org/~acme/bpf/devconf.cz-2020-BPF-The-Status-of-BTF-producers-consumers/#/33

[6] "BPF CO-RE (Compile Once – Run Everywhere)", Andrii Nakryiko
http://vger.kernel.org/bpfconf2019_talks/bpf-core.pdf

[7] "Rethinking bpfilter and user-mode helpers", Jonathan Corbet, June 12, 2020
https://lwn.net/Articles/822744/

[8] "Linux kernel lockdown, integrity, and confidentiality", Matthew Garret, April 21, 2020
https://mjg59.dreamwidth.org/55105.html

[9] "evm: re-release", Mimi Zohar, March 15, 2011
http://git.kernel.org/torvalds/c/66dbc325afce

[10] “bpf: allow loading instructions from a fd”, Matteo Crocce, July 13, 2020
https://lore.kernel.org/bpf/20200713130511.6942-1-mcroce@linux.microsoft.com/

[11] “The Linux Kernel User’s and Administrator’s guide -> Device Mapper -> dm-verity”
https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html

[12] “Integrity Policy Enforcement LSM (IPE)”, Deven Bowers, July 29, 2020

https://lore.kernel.org/linux-audit/20200730003113.2561644-1-deven.desai@linux.microsoft.com/