Contribution ID: **245** Type: **not specified**

Securing trusted boot of confidential VMs

Tuesday, 21 September 2021 10:40 (20 minutes)

Confidential Computing can enable several use-cases which rely on the ability to run computations remotely on sensitive data securely without having to trust the infrastructure provider. One required building block for this is verifiable control flow integrity on the remote machine: ensuring that the running compute is doing what it's supposed to.

With hand-written Intel SGX the code surface is usually limited, but with a fully fledged VM it becomes more difficult. One example for this is securing the control flow of the VM's boot process:

In the last years we have seen multiple projects securing the boot process of confidential VMs (cVMs) by allowing to boot from encrypted disk images. These approaches usually rely on the injection of a secret during the boot process. While this enables use-cases like hiding the raw disk image from the platform provider (i.e. the entity running the hardware and hypervisor stack), we are not capable of creating hardware-backed proofs of the measurement (i.e. hash) of the code (and data) which is being executed on that cVM, also called remote attestation.

Modern x86 extensions (like AMD SEV-SNP, Intel TDX) allow measurement of the initial boot image before VM startup and cryptographically bind this measurement in a remotely verifiable attestation document.
The question that now arises is how much code surface the initial measurement should contain and if existing firmwares/bootloaders can be used securely. Taking AMD SEV-SNP as reference hardware we implemented two working proof-of-concepts:
a) Minimal firmware based on existing software: Here we leveraged the work which has been done on OMVF and grub for enabling booting of encrypted images. In that case the attested firmware only consists of the OMVF firmware and the grub bootloader. We extended grub to perform a measurement of the operating system (OS) image during loading and assert the measurement with the known good value baked into the attested firmware. In our case the verified linux image is a EFI unified kernel image which allows us to cryptographically bind the kernel image as well as the initramfs image, and the kernel command line. This approach adds OMFV and grub to the audit surface and makes it hard to provide control flow guarantees. For example without extra hardening the OMFV's EFI shell or the grub shell can be easily used to load a malicious OS image.

b) Custom firmware with OS embedding: In that case the attested firmware also consists of the entire operating system. Using the rust hypervisor firmware as a basis we added support for linking in a disk blob into the firmware. This is done by a virtual "block device" reading from a known in-memory location. With that we get a single measurement over the entire software stack, including the operating system (and potential applications/data). A caveat here is that so far we rely on an ELF binary to be loaded by the VMM (QEMU) and not a flat rom image. Hence, the attested measurement of the hardware will deviate from the direct hash of the firmware file being loaded, requiring some extra steps to verify the attestation. Also measuring a large firmware might be time consuming on the slow Secure Processor (found in AMD SEV-SNP).

The Microconference topic we are proposing would consist of:

- Explaining the problem statement: Control flow integrity of the VM boot process in a post-attestation world
- Quickly going over the two approaches we have taken
- Discussion of:
  - problems of our work and possible improvements
  - other on-going community efforts and collaboration opportunities
  - remaining attacks to break integrity (e.g. configuration provided by a malicious hypervisor,..)

I agree to abide by the anti-harassment policy

I agree

Primary authors: DEML, Stefan; SLEMMER, Andras (decentriq)

Presenters: DEML, Stefan; SLEMMER, Andras (decentriq)

Session Classification: Confidential Computing MC

Track Classification: Confidential Computing MC