Is OVMF too slow for Serverless Confidential Computing?

Tobin Feldman-Fitzthum

IBM Research

Disclaimers

- I like OVMF
- I don’t like serverless

Motivation

SEVeriFast: Minimizing the root of trust for fast startup of SEV microVMs

Benjamin Holmes*
MIT CSAIL
Cambridge, MA, USA
Vassar College
Poughkeepsie, NY, USA
bcwh@csail.mit.edu

Jason Waterman
Vassar College
Poughkeepsie, NY, USA
jawaterman@vassar.edu

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

Abstract

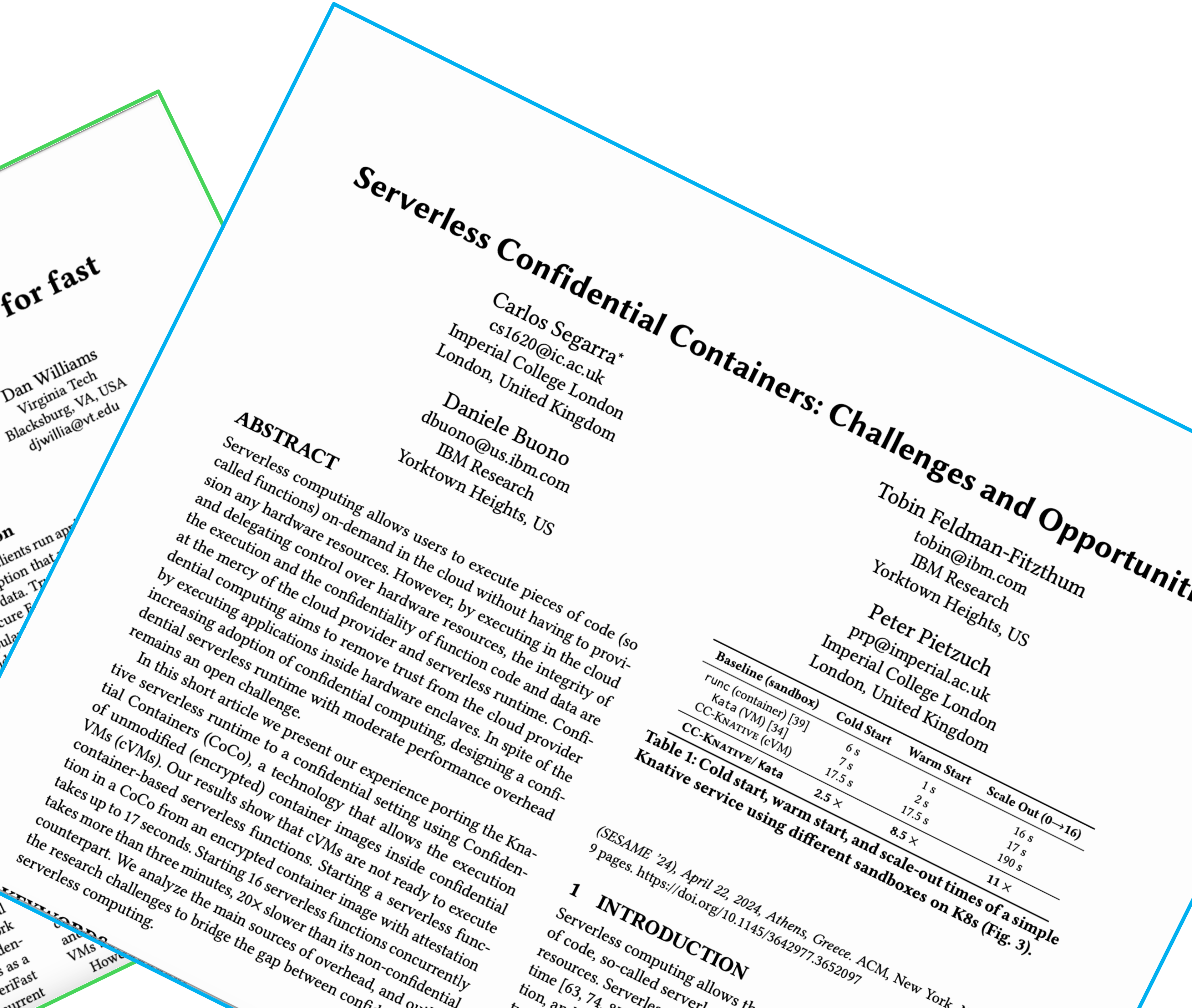
Serverless computing platforms rely on fast container initialization to provide low latency and high throughput for requests. While hardware enforced trusted execution environments (TEEs) have gained popularity, confidential computing has yet to be widely adopted by latency-sensitive platforms due to its additional initialization overhead. We investigate the application of AMD's Secure Encrypted Virtualization (SEV) to microVMs and find that current startup times for confidential VMs are prohibitively slow due to the high cost of establishing a root of trust for each new VM.

We present SEVeriFast, a new bootstrap scheme for SE
VMs that reevaluates current microVM techniques f
boot, such as eliminating bootstrap stages and b
guest kernel decompression. Counter-intuitively,
introducing an additional bootstrap componen
kernel compression optimizes the cold bo
SEV microVMs by reducing the cost of mea
boot path and producing a minimal
SEVeriFast is the first work
with booting confiden-
principles as a
SEVeriFast
current

1 Introduction

In today's cloud, clients run app
with the assumption that
their sensitive data. Tr
like AMD Secure E
become popula
domain and
ning ap
14, 1

and
VMs
How



Serverless Confidential Containers: Challenges and Opportunit

Carlos Segarra*
cs1620@ic.ac.uk
Imperial College London
London, United Kingdom

Tobin Feldman-Fitzthum
tobin@ibm.com
IBM Research
Yorktown Heights, US

Daniele Buono
dbuono@us.ibm.com
IBM Research
Yorktown Heights, US

Peter Pietzuch
prp@imperial.ac.uk
Imperial College London
London, United Kingdom

ABSTRACT

Serverless computing allows users to execute pieces of code (so called functions) on-demand in the cloud without having to provision any hardware resources. However, by executing in the cloud and delegating control over hardware resources, the integrity of the execution and the confidentiality of function code and data are at the mercy of the cloud provider and serverless runtime. Confidential computing aims to remove trust from the cloud provider by executing applications inside hardware enclaves. In spite of the increasing adoption of confidential computing, designing a confidential serverless runtime with moderate performance overhead remains an open challenge.

In this short article we present our experience porting the Knative serverless runtime to a confidential setting using Confidential Containers (CoCo), a technology that allows the execution of unmodified (encrypted) container images inside confidential VMs (cVMs). Our results show that cVMs are not ready to execute container-based serverless functions. Starting a serverless function in a CoCo from an encrypted container image with attestation takes up to 17 seconds. Starting 16 serverless functions concurrently takes more than three minutes, 20× slower than its non-confidential counterpart. We analyze the main sources of overhead, and out the research challenges to bridge the gap between confi serverless computing.

Baseline (sandbox)	Cold Start	Warm Start	Scale Out (0→16)
runc (container) [39]	6 s	1 s	16 s
Kata (VM) [34]	7 s	2 s	17 s
CC-Knative (cVM)	17.5 s	17.5 s	190 s
CC-Knative/ Kata	**2.5 ×**	**8.5 ×**	**11 ×**

Table 1: Cold start, warm start, and scale-out times of a simple Knative service using different sandboxes on K8s (Fig. 3).

(SESAME '24), April 22, 2024, Athens, Greece. ACM, New York, 9 pages. https://doi.org/10.1145/3642977.3652097

1 INTRODUCTION

Serverless computing allows th
of code, so-called server
resources. Serverles
time [63, 74, 8
tion, an

Serverless Confidential Computing

- Warm starts are slower and more complex
- Cold starts are slower and more complex
- Starting a lot of guests at once is slow

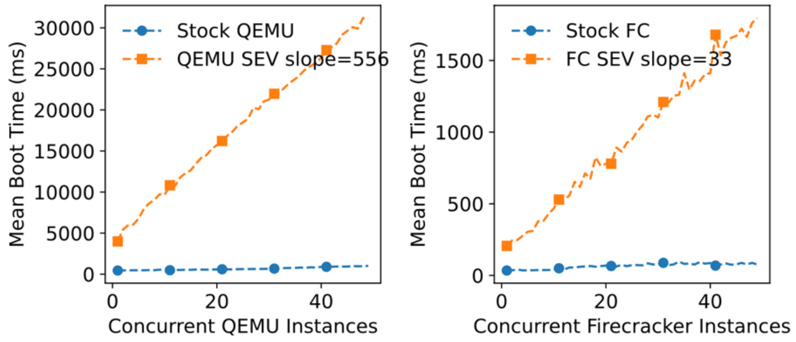
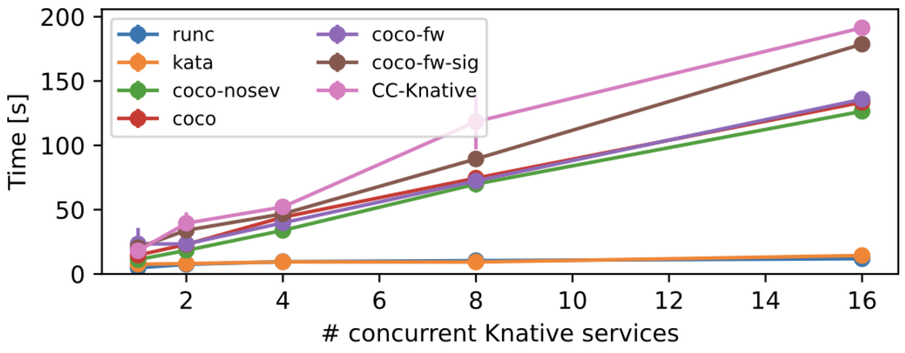


Figure 12. Average boot time of concurrent SEV guests from 1 to 50 concurrent instances.



(b) Throughput-latency of service instantiation.

Cold Starts

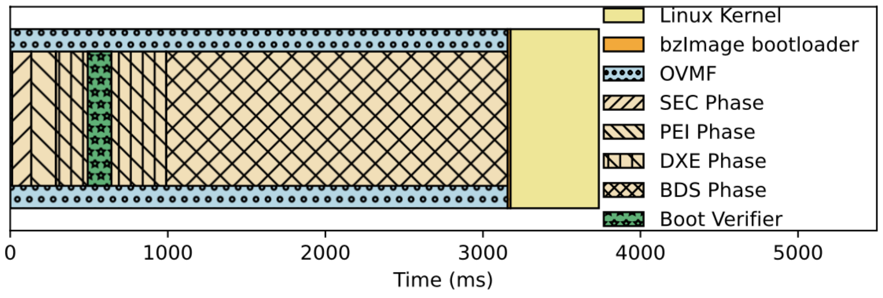


Figure 3. Breaking down the OVMF boot process with SEV-SNP shows that the boot verifier is a small portion of overall boot time.

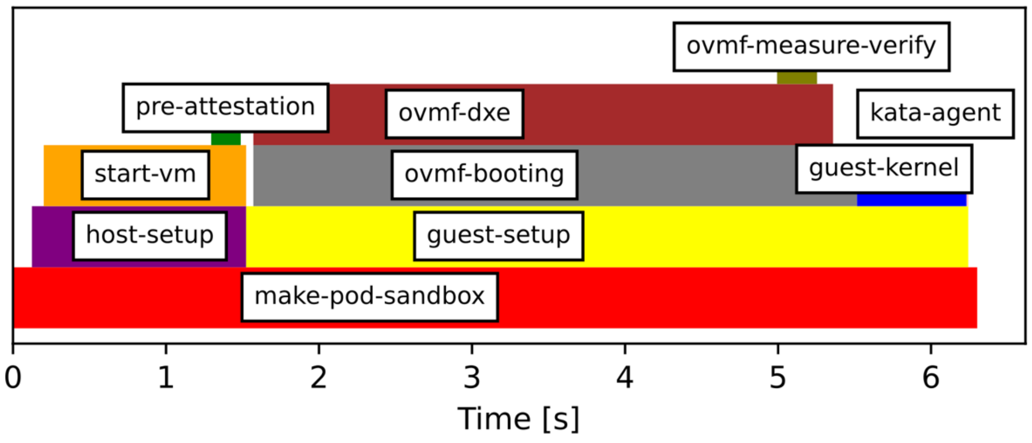
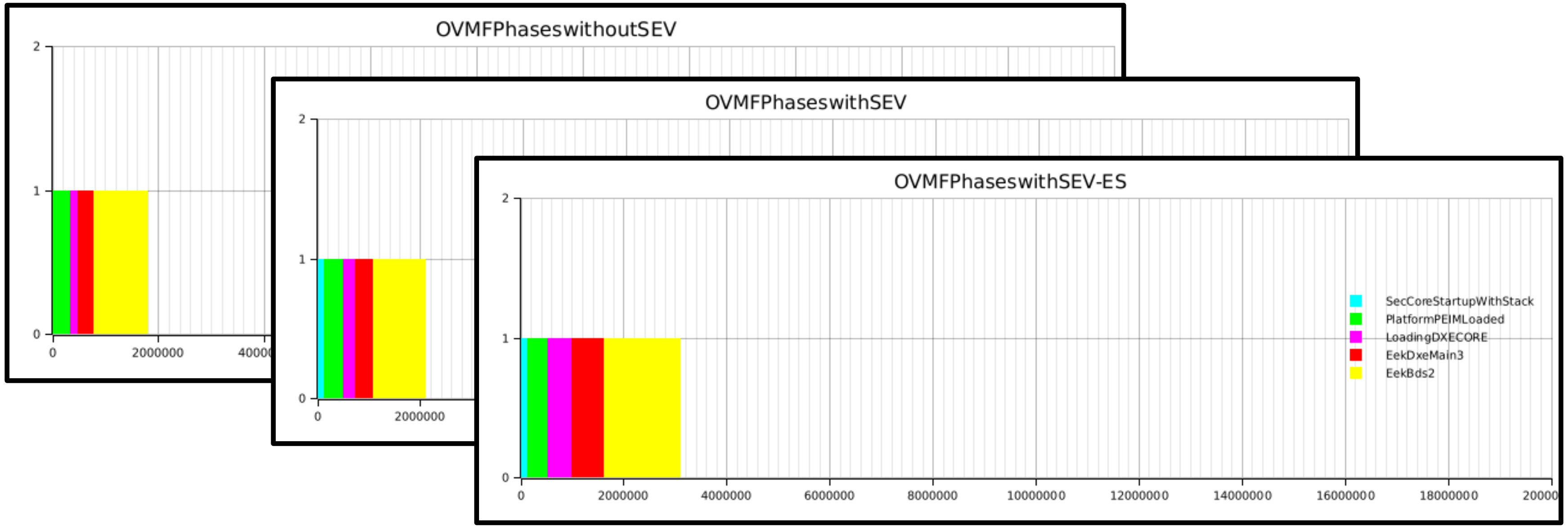


Figure 4: Flame graph of the time spent booting a cVM.

Kernel based on 6.1.0-rc4

- Neither of these papers is primarily focused on OVMF benchmarking
    - Holmes et al care about measurement
- There are lots of different guest configurations and software stacks

Preliminary Results

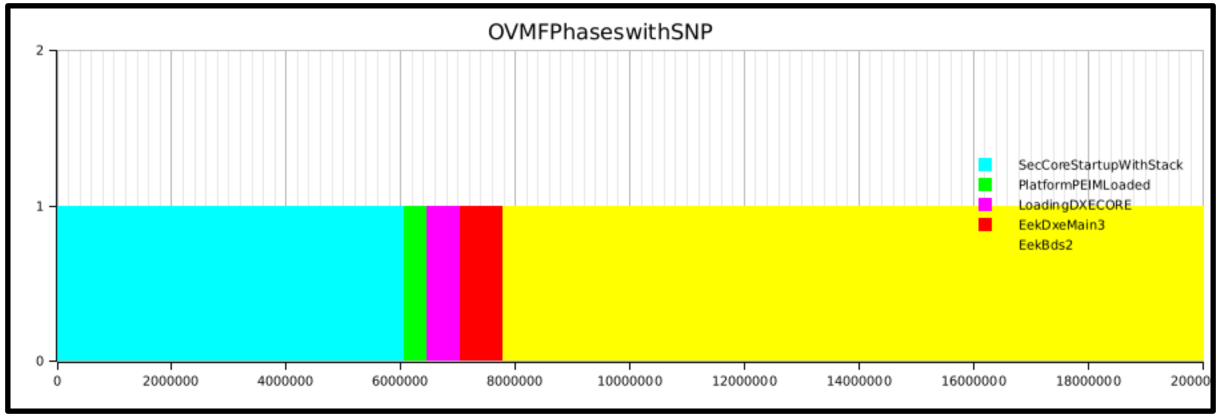


OVMFPhaseswithoutSEV

OVMFPhaseswithSEV

OVMFPhaseswithSEV-ES

Performance Counter Frequency: 3,579,545 hz



OVMFPhaseswithSNP

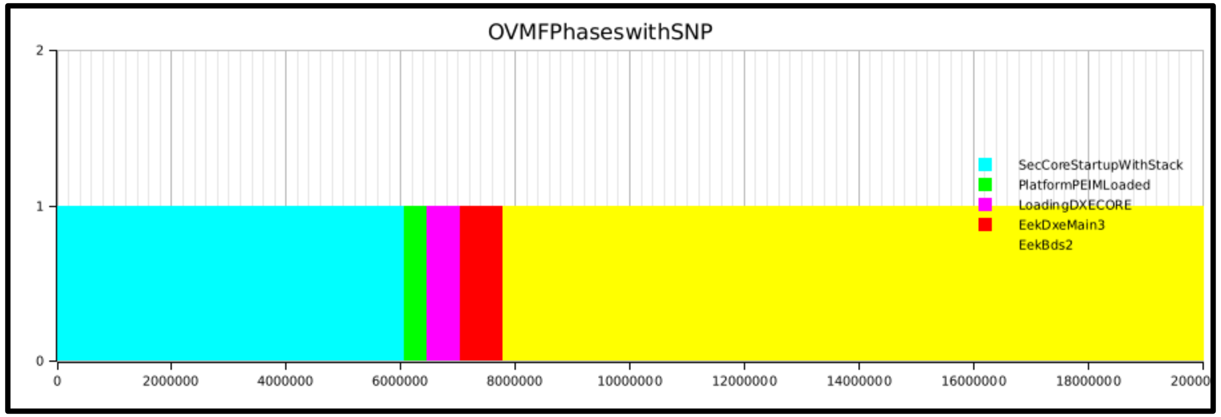
Host kernel is 6.8.0-rc5-next-20240221-snp-host-cc2568386ccb

OVMF Debugging Techniques

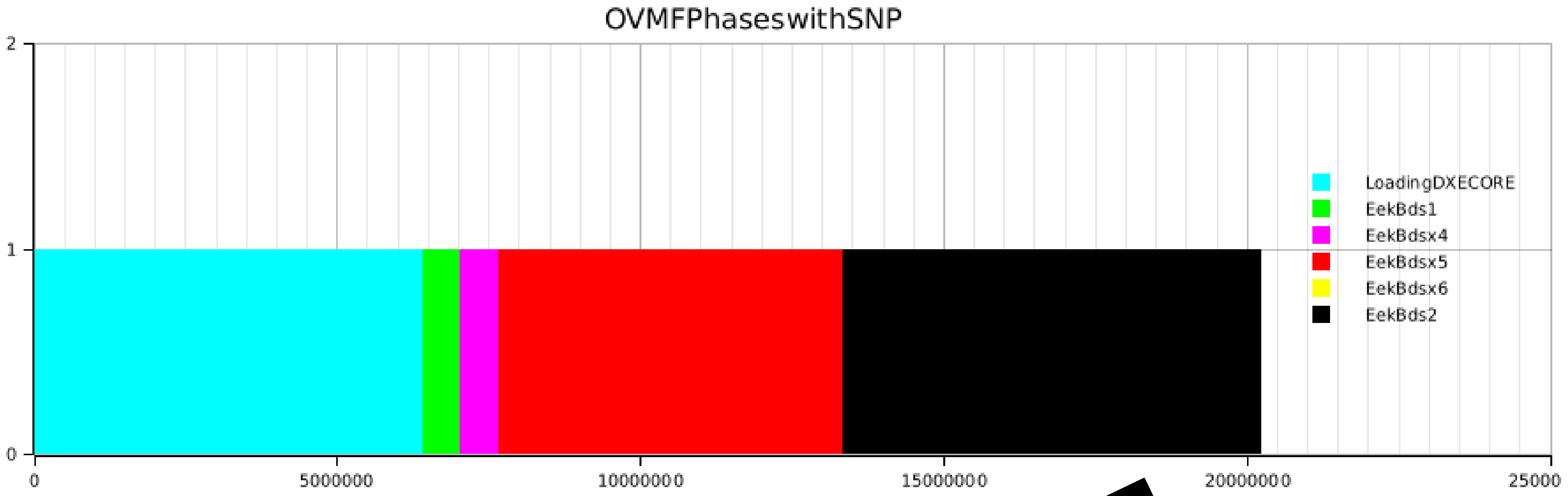
- Debugger
- EFI Profiling
- Printing
    - Watch out for VMExits
- KVM Tracing
- EFI Shell
- Fuzzing
- Read the spec

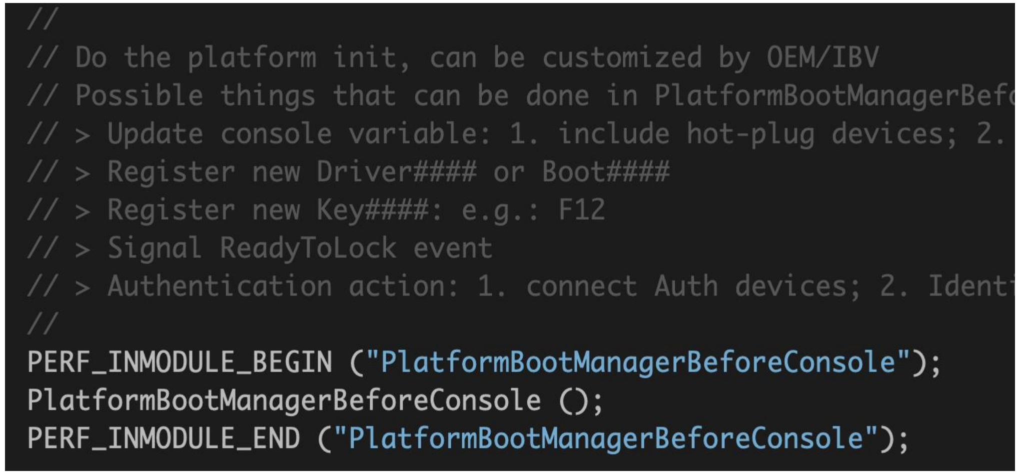
Memory Pre-Validation

- Three places
  - OvmfPkg/Sec/AmdSev.c
  - OvmfPkg/PlatformPei/AmdSev.c
  - OvmfPkg/AmdSevDxe/AmdSevDxe.c
- Plus
  - Unaccepted memory
  - And pre-encryption

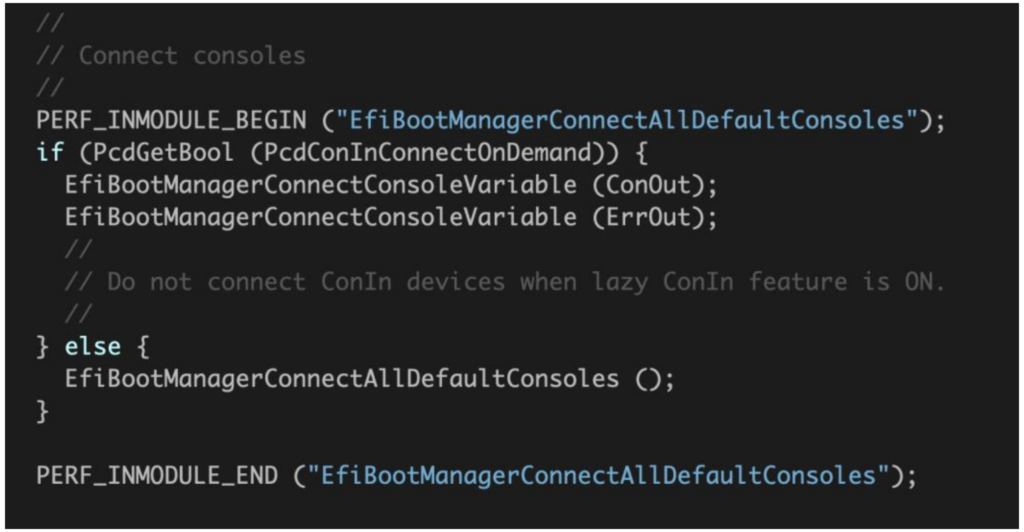


BDS Breakdown





```
//
// Do the platform init, can be customized by OEM/IBV
// Possible things that can be done in PlatformBootManagerBef
// > Update console variable: 1. include hot-plug devices; 2.
// > Register new Driver#### or Boot####
// > Register new Key####: e.g.: F12
// > Signal ReadyToLock event
// > Authentication action: 1. connect Auth devices; 2. Ident
//
PERF_INMODULE_BEGIN ("PlatformBootManagerBeforeConsole");
PlatformBootManagerBeforeConsole ();
PERF_INMODULE_END ("PlatformBootManagerBeforeConsole");
```

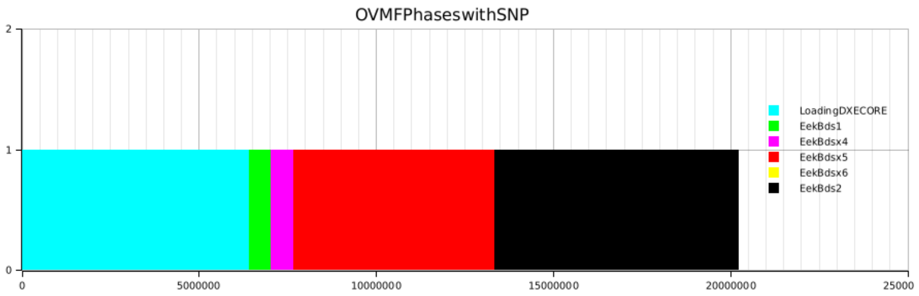


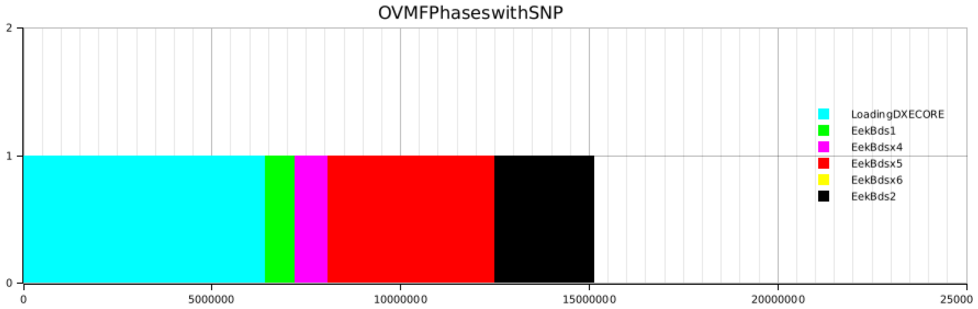
```
//
// Connect consoles
//
PERF_INMODULE_BEGIN ("EfiBootManagerConnectAllDefaultConsoles");
if (PcdGetBool (PcdConInConnectOnDemand)) {
  EfiBootManagerConnectConsoleVariable (ConOut);
  EfiBootManagerConnectConsoleVariable (ErrOut);
  //
  // Do not connect ConIn devices when lazy ConIn feature is ON.
  //
} else {
  EfiBootManagerConnectAllDefaultConsoles ();
}

PERF_INMODULE_END ("EfiBootManagerConnectAllDefaultConsoles");
```

MdeModulePkg/Universal/BdsDxe/BdsEntry.c

Audience Participation

- Which one is slowing us down?
  - **A. -device driver=virtio-net-pci,netdev=network-0,mac=ba:2f:08:16:18:aa,disable-modern=false,mq=on,vectors=4**
  - **B. -numa node,memdev=dimm1**
  - **C. -device virtio-scsi-pci,id=scsi,disable-modern=false**
  - **D. -device virtio-rng-pci,rng=rng0**

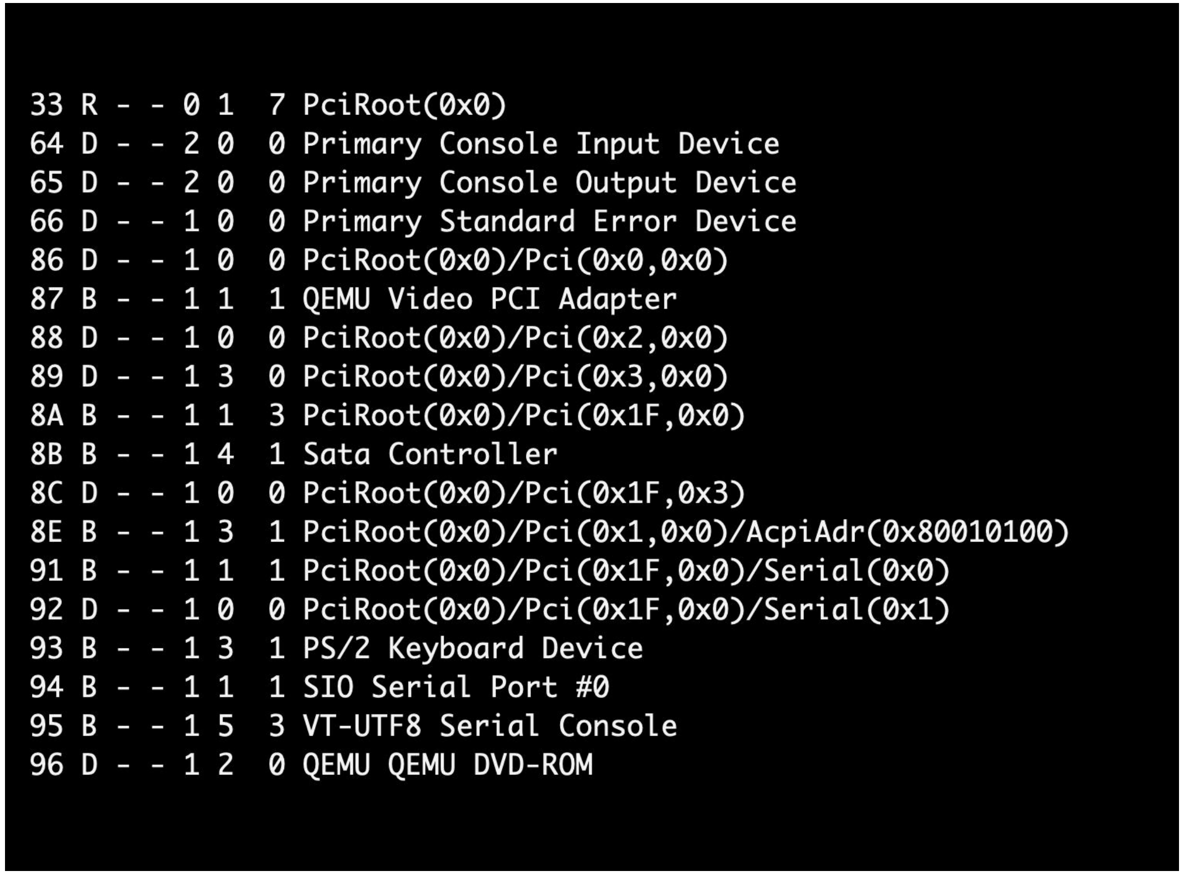




Virtio-Rng??

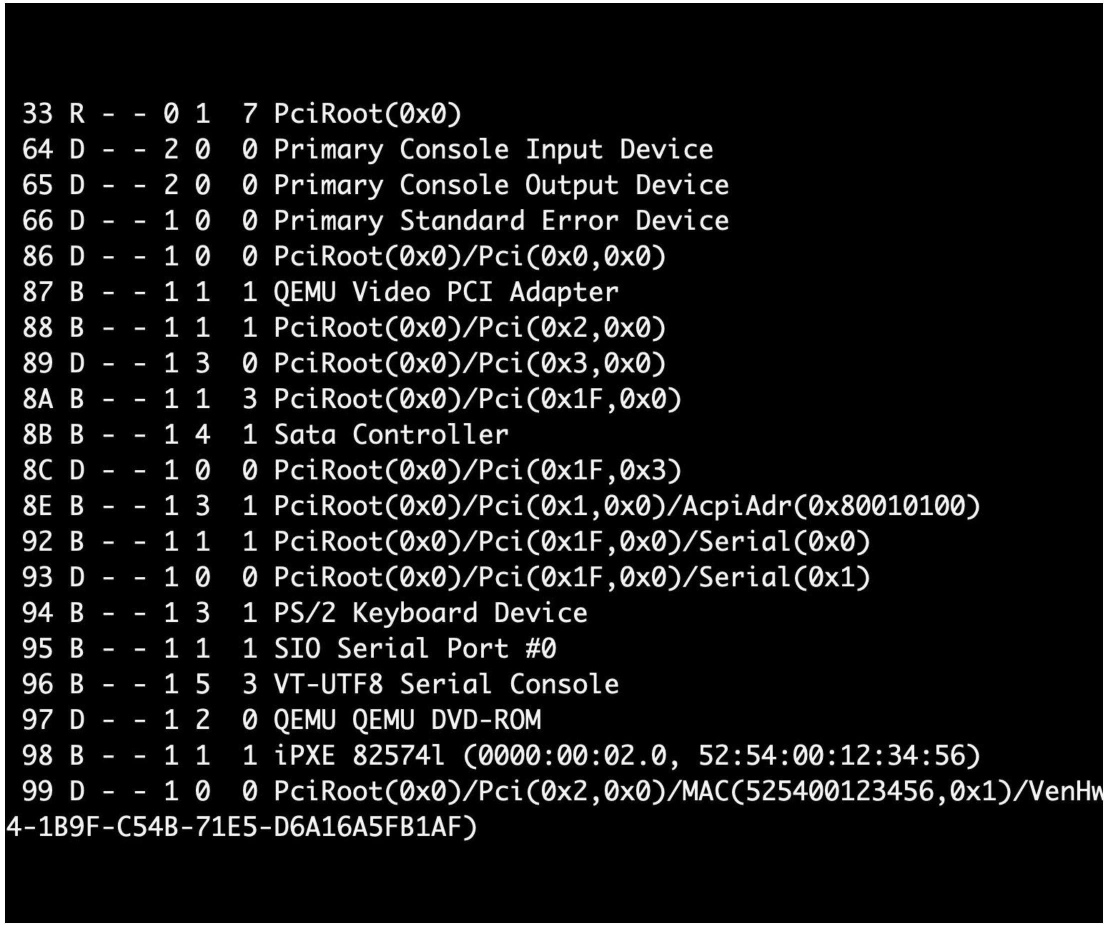
- What does connecting consoles have to do with virtio-rng?
- Connecting consoles is surprisingly complex
  - See 3.15.3
  - Handles, Devices, Device Paths, Drivers
  - OVMF tries to bind most devices to most drivers
  - Could this be more enlightened?
- Why is Virtio-Rng slow?
  - It isn't
  - When it is enabled something else is slow
- Remove virtio-rng from Kata

Something else?



```
33 R - - 0 1  7 PciRoot(0x0)
64 D - - 2 0  0 Primary Console Input Device
65 D - - 2 0  0 Primary Console Output Device
66 D - - 1 0  0 Primary Standard Error Device
86 D - - 1 0  0 PciRoot(0x0)/Pci(0x0,0x0)
87 B - - 1 1  1 QEMU Video PCI Adapter
88 D - - 1 0  0 PciRoot(0x0)/Pci(0x2,0x0)
89 D - - 1 3  0 PciRoot(0x0)/Pci(0x3,0x0)
8A B - - 1 1  3 PciRoot(0x0)/Pci(0x1F,0x0)
8B B - - 1 4  1 Sata Controller
8C D - - 1 0  0 PciRoot(0x0)/Pci(0x1F,0x3)
8E B - - 1 3  1 PciRoot(0x0)/Pci(0x1,0x0)/AcpiAdr(0x80010100)
91 B - - 1 1  1 PciRoot(0x0)/Pci(0x1F,0x0)/Serial(0x0)
92 D - - 1 0  0 PciRoot(0x0)/Pci(0x1F,0x0)/Serial(0x1)
93 B - - 1 3  1 PS/2 Keyboard Device
94 B - - 1 1  1 SIO Serial Port #0
95 B - - 1 5  3 VT-UTF8 Serial Console
96 D - - 1 2  0 QEMU QEMU DVD-ROM
```

Without Virtio-Rng

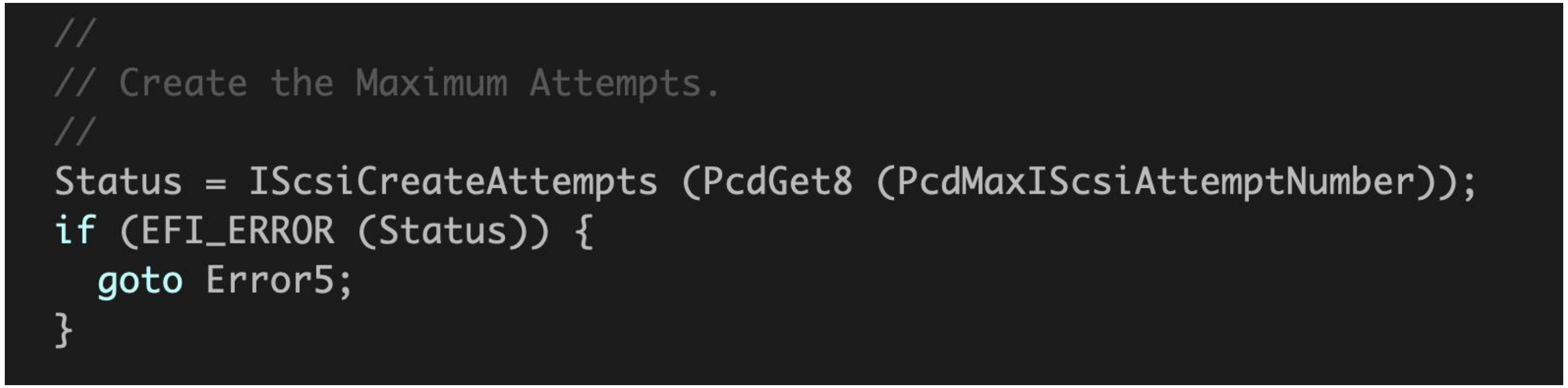


```
 33 R - - 0 1  7 PciRoot(0x0)
 64 D - - 2 0  0 Primary Console Input Device
 65 D - - 2 0  0 Primary Console Output Device
 66 D - - 1 0  0 Primary Standard Error Device
 86 D - - 1 0  0 PciRoot(0x0)/Pci(0x0,0x0)
 87 B - - 1 1  1 QEMU Video PCI Adapter
 88 B - - 1 1  1 PciRoot(0x0)/Pci(0x2,0x0)
 89 D - - 1 3  0 PciRoot(0x0)/Pci(0x3,0x0)
 8A B - - 1 1  3 PciRoot(0x0)/Pci(0x1F,0x0)
 8B B - - 1 4  1 Sata Controller
 8C D - - 1 0  0 PciRoot(0x0)/Pci(0x1F,0x3)
 8E B - - 1 3  1 PciRoot(0x0)/Pci(0x1,0x0)/AcpiAdr(0x80010100)
 92 B - - 1 1  1 PciRoot(0x0)/Pci(0x1F,0x0)/Serial(0x0)
 93 D - - 1 0  0 PciRoot(0x0)/Pci(0x1F,0x0)/Serial(0x1)
 94 B - - 1 3  1 PS/2 Keyboard Device
 95 B - - 1 1  1 SIO Serial Port #0
 96 B - - 1 5  3 VT-UTF8 Serial Console
 97 D - - 1 2  0 QEMU QEMU DVD-ROM
 98 B - - 1 1  1 iPXE 82574l (0000:00:02.0, 52:54:00:12:34:56)
 99 D - - 1 0  0 PciRoot(0x0)/Pci(0x2,0x0)/MAC(525400123456,0x1)/VenHw
4-1B9F-C54B-71E5-D6A16A5FB1AF)
```

With Virtio-Rng

SCSI

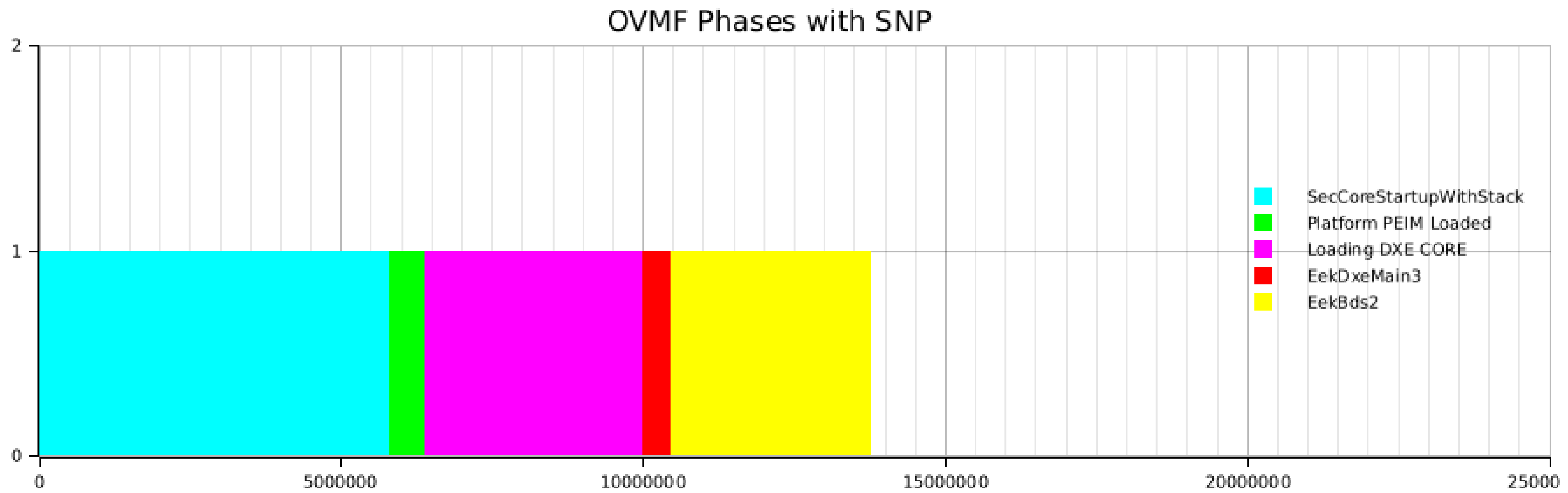
- This is slow



```
//
// Create the Maximum Attempts.
//
Status = IScsiCreateAttempts (PcdGet8 (PcdMaxIScsiAttemptNumber));
if (EFI_ERROR (Status)) {
  goto Error5;
}
```

- Because it sets EFI variables
- On SEV(-ES) initializing QEMU Flash fails, so we avoid the slow path

What about the new kernel?

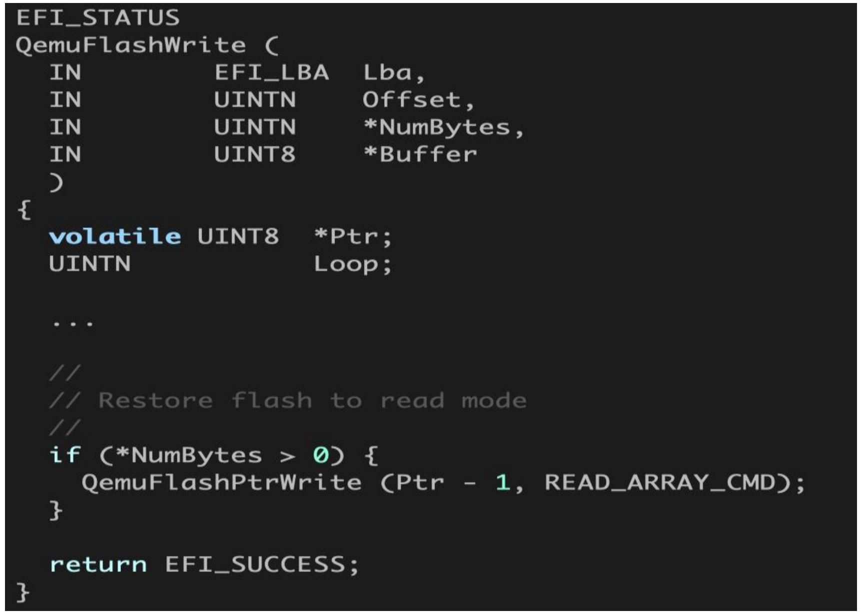


- Writing to Pflash nvdata no longer slow
    - Perhaps because readonly memslots are no longer allowed for SVMs
    - Not the end of the story
- There is some overhead from pvalidate
    - It’s less than expected

(6.11.0-rc5-snp-host-cc2568386)

QEMU Flash

- Why are we emulating flash at all?
  - OVMF doesn’t know how flash is provided
- QEMU allows potentially invalid configurations
- On SEV-ES we don’t use QEMU flash, but it might be a bug



```
EFI_STATUS
QemuFlashWrite (
  IN        EFI_LBA  Lba,
  IN        UINTN    Offset,
  IN        UINTN    *NumBytes,
  IN        UINT8    *Buffer
  )
{
  volatile UINT8  *Ptr;
  UINTN           Loop;

  ...

  //
  // Restore flash to read mode
  //
  if (*NumBytes > 0) {
    QemuFlashPtrWrite (Ptr - 1, READ_ARRAY_CMD);
  }

  return EFI_SUCCESS;
}
```

Example nvdata

Variable NV+RT+BS 'EFIGlobalVariable:BootOrder' DataSize = 0x04
Variable NV+RT+BS 'EFIGlobalVariable:Boot0001' DataSize = 0x58
Variable NV+RT+BS 'EFIGlobalVariable:ErrOut' DataSize = 0x49
Variable NV+RT+BS 'EFIGlobalVariable:ConIn' DataSize = 0x7A
Variable NV+RT+BS 'EFIGlobalVariable:ConOut' DataSize = 0x67
Variable NV+RT+BS 'EFIGlobalVariable:Key0001' DataSize = 0x0E
Variable NV+RT+BS 'EFIGlobalVariable:Key0000' DataSize = 0x0E
Variable NV+RT+BS 'EFIGlobalVariable:Lang' DataSize = 0x04
Variable NV+RT+BS 'EFIGlobalVariable:PlatformLang' DataSize = 0x03
Variable NV+RT+BS 'EFIGlobalVariable:Timeout' DataSize = 0x02
Variable NV+RT+BS 'EFIGlobalVariable:Boot0000' DataSize = 0x3E
Variable NV+RT+BS 'EFIGlobalVariable:Boot0000' DataSize = 0x3E

Conclusion

- OVMF is not inherently slow with Confidential Computing
- But it is complex and difficult to analyze or optimize
  - Many configurations not regularly tested
- Let’s figure out confidential warm starts

- Things to fix
  - Remove virtio-rng from Kata
  - Add QEMU warnings for invalid configurations
  - Take a close look at QEMU Flash and nvdata
  - Figure out SEC overhead