AES Flow Interception : Key Snooping Method on Virtual Machine. - Exception Handling Attack for AES-NI -

Tatsuya TAKEHISA[⋆], Hiroki NOGAWA[⋆⋆], and Masakatu MORII[⋆ ⋆ ⋆]

Abstract. In this paper, we propose a method for snooping AES encryption key on Virtual Machine Monitor (VMM), and we present countermeasures against this attack. Recently, virtualization technology has rapidly emerged as a key technology for cloud computing. In general, the virtualization technology composes two software parts: one is virtual machine (VM) management software called Virtual Machine Monitor (VMM), and the other is its associated software. The virtualization technology at present does not provide methods for certifying dependability of the VMMs. In this situation, the following case is possible: when malicious service providers serve tampered VMMs and their users run their VMs on these VMMs, the users will suffer unintended information leakage. As one leakage case, in this paper, we propose a method for snooping AES encryption key on the VMM. In addition, we present countermeasures against this key snooping.

Keywords: Virtualization, Side Channel Attack, AES-NI, Key Snooping, Exception Handling

1 Introduction

Recently, virtualization technology has emerged for accommodating multiple function units into one computer. In the center of the virtualization technology, software called Virtual Machine Monitor(VMM) controls virtual machines that are virtualized computers and the VMM intercepts virtual machines for system stability. The virtual machines encrypt data inside them for security reason. Among encryption methods, common key encryption methods are used for encryption performance. Among many common key encryption methods, Advanced Encryption Standard (AES)[1] is used in most cases, and AES is implemented as software in some cases, and as hardware in other cases.

Against AES, many attack methods have been proposed. Among these attacks, in this paper, we focus on methods that attack vulnerability of software

[⋆] Japan Datacom, Co., Ltd., Akasaka, Comfy Homes 6B 6-6-28 Akasaka, Minato-ku, Tokyo 107-0052, Japan, takehisa@datacom.co.jp

[⋆⋆] Secureware Inc., Kamishinden 1-74-1-204, Toyonaka City, Osaka, 560—0085 Japan, nogawa@secure-ware.com

[⋆ ⋆ ⋆] Graduate School of Engineering, Kobe University, 1-1, Rokkodai, Nada-ku, Kobe-shi, 657-8501 Japan, mmorii@kobe-u.ac.jp

implementation of AES. Previous papers proposed side channel attack against AES. Side channel attacks proposed are not able to obtain the encryption key itself, and only able to get candidates of the encryption keys more efficiently than brute force attack.

However, our method enables an attacker to guess the encryption key itself when he intercepts a VM by a modified VMM. As a result, the attacker is able to decrypt the secret data and to tamper with it. Our method consists of three stages as follows:

1) An attacker executes a modified VMM on a CPU where Intel AES-NI instruction is not implemented.
2) The attacker intercepts a guest OS with fake return value of CPUID instruction by the modified VMM.
3) When the guest OS executes the AES-NI instruction, the modified VMM generates invalid Opcode Exception. Then, the attacker is able to estimate the encryption key stored in CPU registers or memory at Opcode Exception.

2 Architecture for Key Snooping

In this section, we briefly state architecture where our key snooping method is possible. This section consists of four subsections describing the following components: 1) AES-NI that is a hardware implementation of AES, 2) CPUID instruction that is a basis of our attack method, 3) Method for checking whether application software can call AES-NI, 4) Intel-VT for obtaining the timing of the attack.

2.1 Intel AES New Instruction(AES-NI)

Intel developed AES-NI for rapid processing (by hardware) of the AES algorithm[3]. AES-NI contains some AES sub step components as hardware implementation for rapid processing of AES encryption/decryption. In addition, implementation of AES-NI instructions in hardware prevents software side channel attacks.

AES-NI consists of six instructions that provide complete hardware support for AES. Four instructions support the AES encryption and decryption, and the other two instructions support the AES key expansion. Table 1 shows details of the six instructions of AES-NI.

Figure 1 displays a sample flow of encryption/decryption for AES algorithm (ECB mode) by the AES-NI instructions. For encryption, the AES repeats AESENC instruction between 1 and n-1 and the final stage does AESENCLAST instruction. Similarly, for decryption, the AES repeats AESDEC instruction between 1 and n-1, and the final stage does AESDECCLAST instruction.

AES-NI Instruction	Description
AESENC	Perform One Round of an AES Encryption Flow
AESENCLAST	Perform Last Round of an AES Encryption Flow
AESDEC	Perform One Round of an AES Decryption Flow
AESDECLAST	Perform Last Round of an AES Decryption Flow
AESIMC	Perform the AES InvMixColumn Transformation
AESKEYGENASSIST	AES Round Key Generation Assist

Table 1. AES-NI Instructions

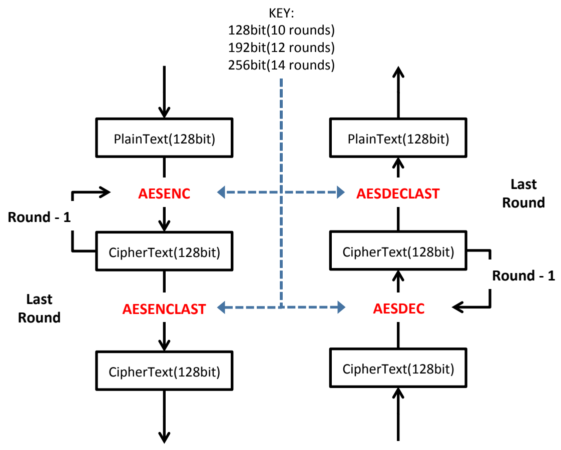

Fig. 1. AES-NI Encription/Decription Flow for AES-ECB mode

2.2 CPUID Instruction

Intel-based CPU contains extended instructions for rapid processing and function extensions. When a CPU performs an extended CPU instruction in a case where the CPU does not implement the instruction, interrupt execution occurs as a general protection error. This is the reason why OS and application software have to implement methods for checking the extended CPU instructions. For checking the extended CPU instructions, OS and application software call the CPUID instruction, the return value of which are four types of information: vender, serial number, enhanced feature and cache information of CPU.

In addition, CPUID instruction returns processor identification and feature information by means of EAX, EBX, ECX and EDX registers. The instruction's output depends on the contents of the EAX register. Table 2 shows arguments and return values of the CPUID instruction.

2.3 How to Check AES-NI support

The following procedure describes a method for checking AES-NI support:

Argument(EAX)	Return values
0	EAX=Maximum Input Value for Basic CPUID Information EBX,ECX,EDX=Vendor ID
1	EAX=Version Information EBX,ECX,EDX=Processor Signature and Feature Bits
2	EAX,EBX,ECX,EDX=Cache and TLB Information
3	EAX,EBX=Reserved ECX,EDX=Processor Serial Number

Table 2. Argument and return value of CPUID

1) Check whether the processor supports SSE/SSE2
   (if CPUID.01H:EDX.SSE[bit 25] = 1)
2) Check whether the processor supports AESNI
   (if CPUID.01H:ECX.AESNI[bit 25] = 1)
3) Check whether the processor supports PCLMULQDQ
   (if CPUID.01H:ECX.PCLMULQDQ[bit 1] = 1)

In most cases, only the second procedure works well.

2.4 Intel VT

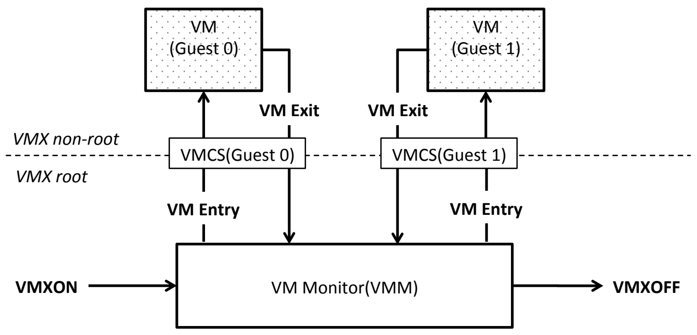

Fig. 2. Interaction of a VMM and Guest OSes (VMs)

Intel released the Intel Virtualization Technology (Intel VT) to assist virtualization with hardware. Intel introduced two new CPU execution modes into Intel-VT: two VMX modes as Virtual Machine Extensions (VMX). One is VMX root mode, and the other is VMX non-root mode. The VMX root mode is an exclusive mode for VMM to work, and the VMX non-root mode is a mode to allow users execute guest OSes. The concept of these VMX modes is different from conventional privilege ring (Ring0 - 3) at Intel x86 architecture. Both these

VMX modes are able to use conventional privilege level so that running the guest OS at privilege level is possible.

Figure 2 displays interaction of VMM and guest OSes on the Intel-VT. For executing a guest OS with a VMX mode, at first VMXON instruction is executed, and then, the VMM runs in the VMX root mode. Then, VMM prepares a structure body (VMCS) to store a state (CPU register, CPU State, etc.) of each VM. When the VMM switches over to VMX non-root mode, it executes a VMRESUME instruction, and this state transition is named as VM Entry. When the guest OS executes privileged instruction and hardware access, the guest OS switches from the VMX non-root mode to the VMX root mode, and this state transition is named as VM exit. At this state transition, The VMM checks operations in the VM exit, and the VMM processes the operations on behalf of the guest OS. In this way, VMM runs multiple guest OSes while it switches between the state of VM Enter and VM Exit.

3 Key Snooping Method

In this section, we present our method for AES key snooping. Our attack consists of two parts: one is to fake the return value of CPUID instruction of the VM, and the other is to estimate an AES key by intercepting exceptions of the VM. Subsection 3.1 describes the first part, and subsection 3.2 shows the second part.

3.1 Faking the CPUID instruction

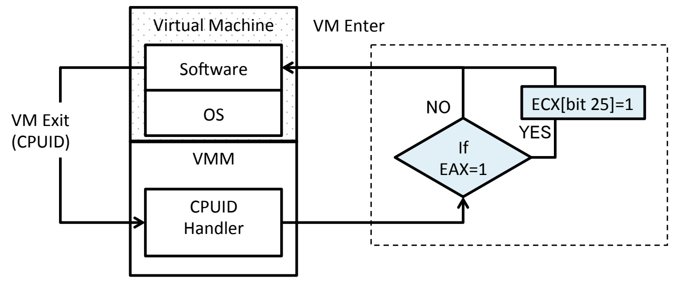

Fig. 3. How to fake CPUID

Figure 3 depicts how to fake CPUID instruction. In fig.3, the VMM catches exceptions occurring in the virtual machine (VM). Exceptions are read/write access to hardware and executing privilege instructions. For secure and robust operation of the VMs, the VMM mediates each exception called by the VMs. This mediation is essential for assuring secure and robust operation of VMs on varying resources of the VMM, as resources of VMM are not always assured. For example, the VMM intercepts the execution of the CPUID instruction to mimic the VM as if is working on a real machine. In addition, the VMM sometimes makes the VM not to use some functions of the CPU while the CPU implements the functions. When the VMM informs fake information of CPU instruction extension to the VM, the VM usually tries to use its instruction extension to increase processing speed. When the CPU receives an unimplemented instruction, the CPU generates an exception called Invalid Opcode instruction. Our attack method changes return value of the CPUID instruction so that AESNI bit (CPUID.01H:ECA.AESIN[bit 25]) is on. This change causes the VM to call exception.

3.2 Exception handling of AES-NI instrcution

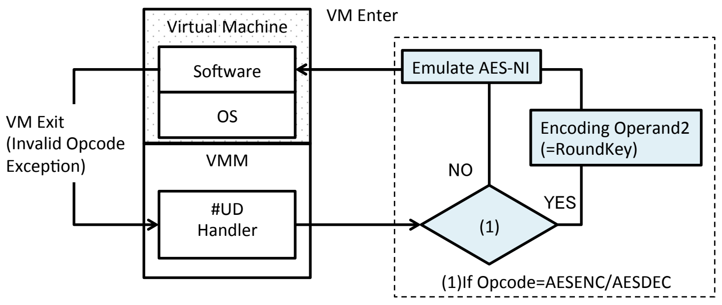
Virtual Machine
Software
OS
VMM
#UD Handler
VM Exit (Invalid Opcode Exception)
VM Enter
Emulate AES-NI
Encoding Operand2 (=RoundKey)
NO
YES
(1)
(1)If Opcode=AESENC/AESDEC

Fig. 4. Exception Handling on VMM

Figure 4 displays our attack method in view of exception handling of AES-NI instruction. Our attack method consists of six steps as follows, from interception of exception in the VMM to AES key estimation:

1) When the VM executes AES-NI instruction, the CPU generates an invalid opcode exception, which causes VM exit state transition. Then, the VMM recovers control.
2) The attacker observes VM exit state transition cause and checks whether invalid opcode instruction is AESENC or AESDEC instruction. Table 3 shows detail of AES-NI instructions, opcode, and operand.
3) If the invalid opcode instruction is AESENC or AESDEC instruction, then the attacker estimates the next operands of the instructiion.
4) The attacker examines the place of stored key from the estimated operands.[1]
5) The attacker estimates AES key or the relevant plaintext from the result of 4), as the keys exists in memory or XMM register on the VM.

[1] As shown in table 3, the first operand is "ModRM:reg(r,w)" and the second operand is " ModRM:r/m(r) ". The attacker is able to estimate the place of stored key in memory or XMM register of the VM from the contents of " ModRM:reg(r,w) ", " ModRM:r/m(r) " [5](Table 2-2)

6) After estimation of the AES key or the relevant plaintext, the attacker emulates AES-NI, not to let the VMM notice failure of AES-NI procedure.[2]

Instruction	Opecode	Operand 1	Operand 2	Operand 3
AESENC	0x66,0x0F,0x38,0xDC	ModRM:reg(r,w)	ModRM:r/m(r)	none
AESENDLAST	0x66,0x0F,0x38,0xDD	ModRM:reg(r,w)	ModRM:r/m(r)	none
AESDEC	0x66,0x0F,0x38,0xDE	ModRM:reg(r,w)	ModRM:r/m(r)	none
AESDECLAST	0x66,0x0F,0x38,0xDF	ModRM:reg(r,w)	ModRM:r/m(r)	none
AESIMC	0x66,0x0F,0x38,0xDB	ModRM:reg(w)	ModRM:r/m(r)	none
AESKEYGENASSIST	0x66,0x0F,0x3A,0xDF	ModRM:reg(w)	ModRM:r/m(r)	imm8

Table 3. AES-NI Instruction, Opecode, Operand

4 Discussion on AES key Snooping

In this section, we discuss essential basis of AES key snooping we have stated, and countermeasures against it. In section 3, we showed our AES key snooping attack that induces exception and does snooping of the AES key or related plaintext when the exception occurs. Here, we define our attack as Exception Handling Attack (EH Attack).

Let us consider the situation where EH attack is possible. The situation is as follows: when a CPU vendor adds new cryptographic instructions and produces new CPUs that implement the new instruction in hardware, a time lag or gap exists until every server installs the new CPUs. A malicious attacker takes advantage of this gap, which means the attacker enjoys EH attack. When an attacker provides massive cloud service over the modified VMM, he obtains every encryption key used in every VM.

Here, we state countermeasures against the EH attack, although these countermeasures do not provide fundamental solutions.

– Suggestion A:(best)
  Users should use CPUs which support AES-NI instruction, where exception handling for AES-NI is impossible.

– Suggestion B:(better)
  Users should not use AES-NI instruction in the OS and an application on VM, where exception handling for AES-NI does not occur in the VMM.

[2] To emulate the AES-NI instruction, see Appendix.

– Suggestion C:(so-so)
  When users execute AES-NI instruction, users should confirm CPU cycles of the AES-NI instruction by RDTSC instruction [3], and should check whether EH attack is not running. [4]

5 Conclusion

In this paper, we have stated a new AES key snooping attack on virtual environment, and presented countermeasures against this attack. Our AES key snooping attack is possible when a virtual machine runs on a CPU, which does not support AES-NI, and the virtual machine monitor is modified so that it fakes return value of CPUID instruction. Our AES key snooping attack runs when AES-NI executes in a virtual machine.

Here, we define our attack as Exception Handling Attack (EH Attack). EH attack is executable in the following situation: 1) when a CPU vendor adds new cryptographic instructions and produces new CPUs that implement the new instruction in hardware, and 2) virtual machines are running on CPUs that do not implement the new cryptographic instructions in hardware. This situation tells that for avoiding EH attacks, users should avoid utilizing the new cryptographic instructions until the new CPUs are available for their virtual machines. This avoidance is also applicable when using IaaS (Infrastructure As A Service) or PaaS (Platform As A Service).

Toward facilitating the development and utilization of cloud services, we strongly believe we have to make research for dependability of virtual machine monitors on which virtual machines run.

References

1. FIPS PUB 197, Advanced Encryption Standard(AES), National Institute of Standards and Technology, Nov. 2001, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
2. Securing the Enterprice with Intel AES-NI, Intel Corporation, Sept. 2010, `http://www.intel.com/Assets/en\_US/PDF/whitepaper/323587.pdf`
3. Intel Advanced Encryption Standard (AES) Instructions Set - Rev 3, Jan. 2010, `http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/`
4. Intel 64 and IA-32 Architectures Software Developer ' s Manual Volume 1:Basic Architecture, Intel Corporation, May 2011, `http://www.intel.com/Assets/PDF/manual/253665.pdf`
5. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 (2A&2B): Instruction Set Reference, A-Z, Intel Corporation, May 2011, `http://www.intel.com/Assets/PDF/manual/325383.pdf`

[3] We assume execution cycles of AES-NI instruction such as AESENC/AESDEC would be larger in two digits during the EH attack.

[4] The return value of RDTSC instruction may be set to a suitable value in a tampered smart VMM.

Appendix: Pseudocode for AES-NI Instruction

```
AESENC xmm1, xmm2/m128
    Tmp = xmm1;
    RoundKey = xmm2/m128;
    Tmp = ShiftRows( Tmp );
    Tmp = SubBytes( Tmp );
    Tmp = MixColumns( Tmp );
    xmm1 = Tmp XOR RoundKey;

AESENCLAST xmm1, xmm2/m128
    Tmp = xmm1;
    RoundKey = xmm2/m128;
    Tmp = ShiftRows( Tmp );
    Tmp = SubBytes( Tmp );
    xmm1 = Tmp XOR RoundKey;

AESDEC xmm1, xmm2/m128
    Tmp = xmm1;
    RoundKey = xmm2/m128;
    Tmp = InvShiftRows( Tmp );
    Tmp = InvSubBytes( Tmp );
    Tmp = InvMixColumns( Tmp );
    xmm1 = Tmp XOR RoundKey;

AESDECLAST xmm1, xmm2/m128
    Tmp = xmm1;
    RoundKey = xmm2/m128;
    Tmp = InvShiftRows( Tmp );
    Tmp = InvSubBytes( Tmp );
    xmm1 = Tmp XOR RoundKey;

AESIMC xmm1, xmm2/m128
    xmm1 = InvMixColumns( xmm2/m128 );

AESKEYGENASSIST xmm1, xmm2/m128, imm8
    Tmp3[31:0] = xmm2/m128[127: 96];
    Tmp2[31:0] = xmm2/m128[ 95: 64];
    Tmp1[31:0] = xmm2/m128[ 63: 32];
    Tmp0[31:0] = xmm2/m128[ 31:  0];
    RCON[31:0] = ZeroExtend( imm8[7:0] );
    xmm1[ 31: 0] = SubWord( Tmp1 );
    xmm1[ 63:32] = RotWord( SubWord( Tmp1 ) ) XOR RCON;
    xmm1[ 95:64] = SubWord( Tmp2 );
    xmm1[127:96] = RotWord( SubWord( Tmp3 ) ) XOR RCON;
```