Virtualizing Running System

בתרגיל הקודם הגענו לנקודה בה אנחנו מסוגלים להריץ ב VM קוד מאוד מוגבל שבזיכרון לבחירנו.
כעת, אנחנו נגרום ל Hypervisor להריץ כ VM את המכונה שממנה התקנו את ה hypervisor, כלומר המחשב עצמו ימשיך לרוץ כרגיל אבל מתחת לרגליים שלו ירוץ ה Hypervisor שלנו ויקבל נוטיפיקציות.

APP

נבדוק ש VTX נתמך במעבד שלנו, ניצור את ה device על מנת לטרגר את תהליך האתחול שבעצם יריץ את המחשב שלנו מעל ה Hypervisor, ולבסוף נצא מהתכנית מה שיגרור את סגירת ה device ולסיום הוירטואליזציה שלנו.

DRIVER

כפי שניתן לראות הזזנו חזרה את הלוגיקה העיקרית ל DrvCreate, שם כפי שכבר ראינו (רק באופן קצת יותר מסודר).
קוראים ל initialize_EPTP ול initialize_VMX (שבין היתר גם מקצה זיכרון ל VMCS ול VMXON Region), לאחר מכן עבור כל מעבד מקצים VMM Stack ו MSR Bitmap וקוראים ל VMXSaveState.
ב DrvClose עבור כל מעבד משחררים משאבים שהקצנו, ולאחר מכן רוצים לסיים את הוירטואליזציה - אבל איך ניתן לעשות זאת ? אנחנו רצים בתוך VM ולא יכולים להריץ VMXOFF. הטריק שבחרנו פה הוא לקרוא ל CPUID עם magic שה VMM יוכל לזהות וכאשר יקפוץ VM-Exit נריץ VMXOFF.

VMXSaveState

ראשית, נזכור שהמטרה שלנו היא להכניס את כל הקוד הנוכחי שרץ ל VM, וזה בדיוק מה שהפונקציה הזאת עושה.
היינו רוצים שעבור כל מעבד נשמור את המצב הקיים שלו לפני הקריאה ל VMXSaveState, נריץ את כל ה setup של הוירטואליזציה, ונריץ VMLAUNCH כך שה VM ימשיך לרוץ בדיוק מאותה נקודה שעצרנו (אחרי הקריאה ל VMXSaveState) ועם אותו מצב של מעבד. בעצם פונקציה זו היא "הכנס למצב VM עם הקוד הנוכחי מ ע כ ש י ו !"
נשים לב שזה בדיוק מה שהפונקציה עושה, שומרת את כל האוגרים הרלוונטים (שלא נשמרים ב VMCS), וקוראת לפונקציה VirtualizeCurrentSystem עם ה stack הנוכחי שהוא בעצם ה stack של ה guest...

בהמשך נראה שנגרום להתחלת הריצה ב VM להמשיך בדיוק מאותה נקודה עם ה stack ששלחנו, ועם האוגרים ששמרנו בעזרת הפונקציה VMXRestoreState.

VirtualizeCurrentSystem

ניתן לראות שעושים את כל הדברים ללא שינוי כפי שכבר אינו - VMCLEAR, VMPTRLD עדכון VMCS ו VMLAUNCH.
אך כעת, עדכון ה VMCS נעשה בצורה מעט שונה בעזרת הפונקציה Setup_VMCS_Virtualizing_Current_Machine.

Setup_VMCS_Virtualizing_Current_Machine

האתחול נשאר זהה (שכן בחלק הקודם אתחלנו את האוגרים וכו' עם מידע של ה host) פרט לנקודות הבאות:

- הוספה של תמיכה ב MSR Bitmap filter (יוסבר המשך).
- איפשור ריצה של הפקודות rdtscp, invpcid, xsave, xrestore כיוון שמסתבר ש windows צריך אותם.
- אתחול ה RIP של ה guest להיות VMXRestoreState (זוכרים ? שחזור מצב אחרי כניסה למצב VM) כמובן שגם את ה RSP אתחלנו עם ה RSP עם ה stack שנשלח מ VMXSaveState.

VMExitHandler

כפי שכבר ראינו שמירת מצב וקריאה ל MainVMExitHandler, אך הוספנו קטע שקוד שבודק האם ערך החזרה הוא 1 אז יש לצאת בעזרת VMXOFF ולשחזר RIP + RSP של הפקודה הבאה לביצוע ב guest.

כעת נתאר את ההתייחסויות השונות ל VM-Exit-ים שקופצים:

Cupid

תבין לבד :)

Control Registers Access

קבלת נוטיפיקציה על גישה ל control registers - יש לזה השפעה אדירה במושגי אבטחה (CR4.SMEP\.WP).
עבור כל VM-Exit שונה ניתן לחלץ משדה ה ExitQualification מידע רלוונטי על הפעולה.

Table 27-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Reserved (cleared to 0)
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Reserved (cleared to 0)
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

בקוד שלנו, נחזיר פשוט את האוגרים של ה guest במידה וקוראים אותם, ונעדכן את האוגרים של ה guest במידה וכותבים אליהם. (כמובן שאפשר לעשות פה הרבה דברים מגניבים)

MSR Bitmaps

בגדול מאפשר קבלת נוטיפיקציה על READ או WRITE של MSRs ספציפיים שסימנו ב MSR Bitmap.
ה Bitmap בעצם מורכב מ 4 Bitmaps של READ\WRITE HIGH\LOW MSRs, ניתן להתשמש בפונקציה SetMSRBitmap על מנת לאתחל את ה Bitmap באופן פשוט (המימוש פחות מעניין). נקודה שחשוב לציין שאם לא היינו משתמשים ב MSR Bitmap (שהודענו שאנחנו משתמשים בו באתחול VMCS) היינו מקבלים VM-Exit עבור כל גישה ל MSR מה שמאט את המערכת באופן משמעותי.

כעת, הריצו ובדקו שלא מקבלים BSOD, ושמקבלים את ההודעות דיבוג המתאימות...
יש לנו VM שמריץ בעצם את המחשב עצמו ויוצא ל VMM רק במקרים מסוימים - די מגניב