Hypervisor Magic

בתרגיל הקודם הצלחנו לשים hook על דף של כתובת קרנלית באופן חד-פעמי. כעת נשפר את תשתית ה hooks שלנו, נזריק events, נשתמש ב VPID, נתקן את ה design של הקוד ועוד דברים של סאחים !
הפעם אשתדל לתאר את הקוד בגדול, ואת המודולים הקיימים בו - מקווה שלמדתם כבר לקרוא קוד.

APP

המון שיט של logging שמתקשר בעזרת IOCTLs, מבחינת לב הלוגיקה לא השתנה דבר, בדיקה של תמיכה ב VMX, פתיחה של ה device וסגירה שלו על מנת לסיים את הוירטואליזציה תוך כדי הדפסת לוגים על הדרך.

DRIVER

בחלק זה הקוד כבר נראה יותר נורמלי ודי מופרד למודולים, לכן נפרט על כל מודול \ קונספט בנפרד.

Old Code

כל החלקים הישנים של הקוד של אתחול, כיבוי ה hypervisor די לא השתנו (עד כדי מה שאציין בהמשך + design), ולכן לא אתעכב עליהם פה - מוזמנים לקרוא את החלקים הקודמים :)

Logging

המון השקעה בתשתית Logging ו WPP tracing שזה טוב, אבל לא מעניין אף אחד (מה רע ב import logging ?!).
בכ"מ, לא אתעכב על המימוש שלה ואתם מוזמנים להתעמק אם אתם מרגישים צורך עז להתנבז. (logging.c)
שימו לב למאקרוים הדי שימושיים LogInfo, LogError וכו'.
אציין שנעשה שימוש ב spinlock שמומש לוקלית על מנת לסנכרן אירועים ב VMX root, מהסיבה שסנכרון אירועים ב WinAPI מסתך על IRQL מסוים, וב VMX root אין לך אחד כזה מוגדר וזה מקשה על הסנכרון.

PoolManager

מודול שמספק API די נחמד של הקצאות מ Non-Paged Pool, גם למימוש שלו לא אכנס כי זה די לא מעניין אותנו.
אציין רק שהפונקציה PoolManagerRequestPool ב PoolManager.c פותרת לנו את בעיית ההקצאות מ non-root.

Event Injection

הזרקת Interrupts + Exceptions (מצורף מסמך הסבר הבדלים) ל VM שלנו בעזרת שינוי שדות ב VMCS VM-Entry:

Table 24-14. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT *n*) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

מוזמנים להסתכל על הקובץ events.c ובקוד עצמו על מנת לראות איך מממשים תפיסת והזרקת BP (int 3).

MTF

יכולת הרצת פקודה אחת ב VM וקבלת שליטה חזרה ב VMM. מימוש בפונקציה HvSetMonitorTrapFlag.

Hidden Hooks

נרצה להשתמש ב API ע"י EptPageHook, HvPerformPageUnHookSinglePage, HvPerformPageUnHookAllPages.
השלם את הקוד במקומות הרלוונטים על מנת שה hidden hooks יעבדו.

Read/Write Hooks

תרגיל

Execute Hooks

תרגיל

Removing Hooks

תרגיל

Syscall Hook

עומדות בפנינו מספר אפשרויות למימוש system calls hook:

1. LSTAR MSR - ה handler של הפקודות sysenter\syscall (תכלס KiSystemCall64), ניתן לערוך אותו ובכך לקבל נוטיפיקציה על כל syscall. כמובן ש PatchGuard יכעס ולכן ניתן לעשות את זה בעזרת ה Hypervisor ע"י ניטור MSRs, עדיין זה יהיה מעצבן בגלל תיקונים של Meltdown (מוזמנים לקרוא בזמנכם הפנוי)
2. לשנות את ה SSDT מה ש PatchGuard ישנא (הוא די hater), ולכן ניתן לעשות read hook ולהחזיר תשובה מפוברקת שלנו רק ש KiSystemCall64 קורא הערך, רבל זה די מעצבן ומאוד איטי.
3. תמיכה של VT-x ב disable system call, שיניב exception כל פעם שעושים syscall אנחנו נתפוס אותו בעזרת Exception Bitmap ונממש את הלוגיקה שלנו, די באסה.
4. לעשות Hook מאוד ספיציפי על הפונקציה שאנחנו רוצים (על מה שה SSDT מצביע) בעזרת התשתית שלנו ! עדיין די באסה למצוא את הכתובת של ה system call שהאינדקס שלה ב SSDT משתנה די הרבה אבל עדיין.

מוזמנים לקרוא את הקוד המאוד שביר שב SyscallHook.c .

VPID

על מנת להתשמש ב feature הזה, יש לסמן ב VMCS שמתשמשים בו ובנוסף לתת VPID ייחודי.
מוזמנים להסתכל על vpid.c על מנת לראות דוגמאות שימוש.

Hyper-V Nested Virtualization

MS Hyper-V תומך ב nested-virtualization ולכן אנו יכולים לרוץ מעליו, אך הוא לא "שקוף" לנו ולכן יש לעשות כמה שינויים בקוד כדי לתמוך בו. כאשר אנו רצים מעל Hyper-V הוא בעצם הראשון שיקבל את ה VM-Exit והוא בתורו יקרא ל VM-Exit שלנו, וכאשר נריץ VMRESUME הוא יקבל את זה ובעצם הוא זה שימשיך את ריצת ה VM. הארכיטקטורה הזאת גורמת לכך שאנחנו לא הראשונים לקבל את אירועי ה VM, אך אנו הראשונים לטפל בהם. הבעיה היא ש Windows kernel וה Hyper-V מאוד קשורים בכך שמ"ה מצפה לקבל שירות ע"י VMCALLs ו MSRs מה Hyper-V.
לכן, מכיוון שאנו ראשונים לטפל ב VM-Exit עלינו לבדוק אם הבקשות שקיבלנו הן עבור ה Hyper-V או אלינו.
יש המון פונקציונליות שיש לתמוך בה כדי לממש hypervisor מושלם מעל Hyper-V, אנו לא נממש את כולה אך מוזמנים לקרוא על Hyper-V Hypervisor Top-Level Functional Specification (TLFS) (מצורף מסמך שלא ברור אסהוא מתוקף).

Out of Range MSRs

Windows מעבירים מידע ל VMX root דרך אוגרי MSR ייעודים שהוגדרו ל hypervisor, ומכיוון שהם לא בטווח שהוגדר ל MSR Bitmap, אנו נוסיף תמיכה בקוד שלנו של קריאה גם מהם (על מנת ש Hyper-V יוכל גם לתפוס את זה).

Hyper-V Hypercalls (VMCALLs)

כפי שהזכרנו במילה בעבר כאשר מריצים VMCALL מ root mode זה גורר קריאה ל SMM Monitor, אך כעת זה בעצם יגרור VM-Exit ב Hyper-V ע"פ הקונבציה הבאה:

Register mapping for hypercall inputs when the Fast flag is zero:

x64	x86	Contents
RCX	EDX:EAX	Hypercall Input Value
RDX	EBX:ECX	Input Parameters **GPA**
R8	EDI:ESI	Output Parameters **GPA**

הבעיה היא שצריך לתמוך בקריאה ל Hyper-V כאשר windows עושים את זה, לכן בפונקציה שלנו שיוזמת VMCALL שמנו MAGIC באוגרים שאותם בדקנו ב VMCALL handler ופעלנו בהתאם.

Hyper-V Interface CPUID Leaves

אחד מערכי CPUID ש Hyper-V דורשים שנממש ע"פ TLFS הוא 0x40000001 עבורו יש להחזיר Hv#1 אך ורק במידה ואנו עומדים בכל הדרישות שלהם (ואנחנו לא), לכן נחזיר ערך אחר למשל Hv#0.

כעת, הריצו ובדקו שלא מקבלים BSOD, ושמקבלים את ההודעות דיבוג המתאימות...
עכשיו אנחנו כבר די טובים בוירטואליזציה, ויש עוד הרבה מה ללמוד אבל נגמר :)