digital whisper
Technologic Papers

# Ogres are like onion, it's because they have LAYERS

מאת שי גילת

## הקדמה

אחד הנושאים הכי מוכרים שמדברים על מערכות הפעלה הם privilege rings. למי שלא מכיר את הנושא, ניתן להגדיר שכל מערכת מודרנית פועלת במספר מצבים (או יותר ספציפי - המעבד המודרני), בעלי הרשאות פעילות שונות ובכך השפעות שונות על המערכת ככלל.

כידוע, מערכות ישנות פעלו בצורה הכי פשוטה שיש - לכל תהליך ואובייקט שקיים ופועל על המערכת יש גישה לכל המשאבים על המערכת, ובמובן של תהליכים שרצים, זה כמובן אומר שכל זכרון המערכת היה משותף בין תהליך לתהליך, והכל היה קיים ממש על הזכרון הפיזי.

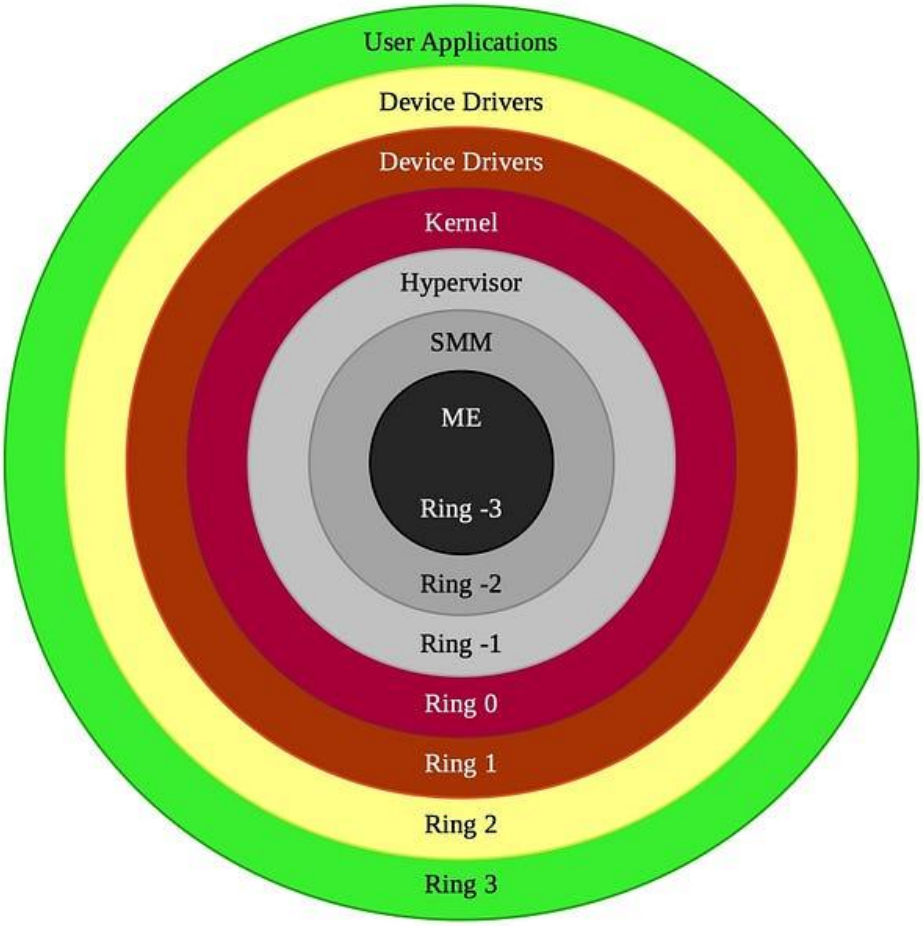
זה אומר שתהליך רגיל היה יכול לשנות את מרחב הכתובות של כל תהליך אחר ואפילו של מערכת ההפעלה עצמה, מה שגרם לבעיות אבטחה קשות מאוד.

בהמשך לתהליך ה-paging, שבא לעזור לנהל זכרון במערכות מודרניות בעזרת פיצול של כל הזכרון הפיזי לחלקים בגודל קבוע ושמירתם בטבלת ניהול הנקראת page table, נוספו גם תיעודים רבים לגבי אותו איזור זכרון שנמצא תחת הדף, בין היתר הרשאות הקוד שיכול לגשת אליו, ככה שקוד של תהליך אחד לא יוכל להשפיע על התהליך האחר שרץ במערכת או בכלל על מערכת ההפעלה.

עם זה התווספו 2 נקודות מרכזיות:

1. תהליך יכול לגשת רק לדפים שממופים לזכרון שלו, ומרחב הכתובות הוירטואליות שלו ממופה לכתובות פיזיות כך שהוא לא יכול בשום צורה להשפיע על תהליך רגיל אחר.
2. נוספה הפרדה בסיסית בין זכרון ברמת הרשאות שונות, כלומר תהליך שרץ ב-ring 3 לא יכול להשפיע על דף זכרון של ring 0 או מתחת.

כך זה נראה כעת:




[מקור: https://medium.com/swlh/negative-rings-in-intel-architecture-the-security-threats-youve-probably-never-heard-of-d725a4b6f831]

## אז מה הנקודה?

כך העקרון השני ממשיך - תהליך / קטע קוד שרץ במעגל אבטחה נמוך יותר, במהות שלו חסין מהשפעות על ידי קוד שרץ במעגל אבטחה גבוה יותר. מהסיבה הזאת, לקוד במעגל אבטחה נמוך יותר יש שליטה גדולה יותר על המערכת (קוד במעגל אבטחה נמוך יותר יכול להשפיע על קוד במעגל אבטחה גבוה יותר, כמו לצפות במישהו מאחורי מראה חד כיוונית - צד אחד יכול לראות לגמרי את הצד השני כמו חלון, אך הצד האחר לא יכול לראות כלום חוץ מההשתקפות של עצמו).

מהסיבה הזאת יש למעגלי האבטחה חשיבות רבה עבור העוסקים בתחום אבטחת המידע, בין אם מכיוון הגנתי או מכיוון התקפי. לכן החלטתי במאמר הזה לתעד ולהסביר על כל מעגל אבטחה ומה שחשוב לדעת עליו, בנוסף לדוגמאות בסיסיות על צורת הרצת קוד באותו מעגל הרשאה.

### מעגל אבטחה 3 - usermode / userspace, איפה שהרגילים נמצאים:

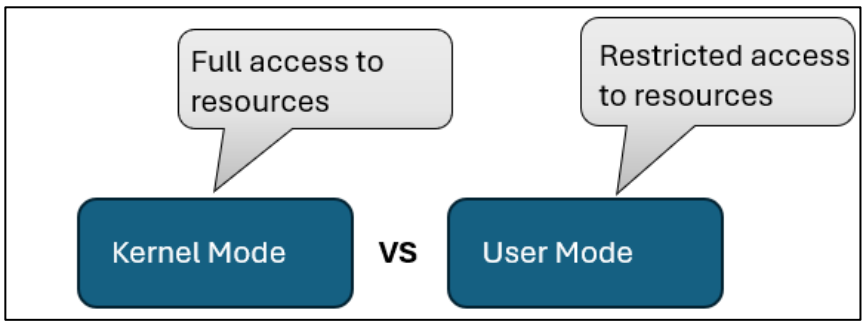
מעגל האבטחה הזה מאפיין את כל התוכנות הרגילות שרצות על המחשב, כל אלו שיש להן graphical user interface ואפשר לעשות איתם אינטראקציה רגילה, ולכן אותן תוכנות מוגבלות הכי הרבה:

1. תוכנות במעגל ריצה 3 מוגבלות אך ורק למרחב הכתובות שלהם: בעזרת מנגנון ה-paging וכתובות וירטואליות, התהליכים הללו חושבים שיש להם גישה לכל הזכרון האפשרי לייצוג כאילו היו במעגל גבוה יותר, אך הם בפועל משתמשים בכתובות וירטואליות שממופות אחרי זה לדף זכרון ספציפי שניתן רק לתהליך הזה, אם בכלל ניתן (תהליך במעגל ריצה 3 יקבל זכרון פיזי רק מתי שיהיה חייב ולא מתי שמבקש, כך שמערכת ההפעלה יוצרת יקום נפרד לתהליך בו הוא חושב שהוא בשליטה מלאה).

2. מהסיבה שמערכת ההפעלה שולטת לגמרי על הכתיבה/קריאה של תהליך במעגל 3, הוא גם לא יכול לכתוב לתוך pages של תהליך אחר במעגל ריצה 3, או כמובן במעגל ריצה נמוך יותר. מערכת ההפעלה מוודאת למפות דפי זכרון בצורה אישית עבור כל תהליך, ומוודאת ששום תהליך רגיל לא יכול לכתוב לזכרון שלא שלו.

3. הגישה שלו למשאבי מערכת מוגבלת על ידי ה-process manager, בין אם זה זכרון פיזי שיכול להשתמש בו, כמות זמן ריצה במעבד או משאבים נוספים.

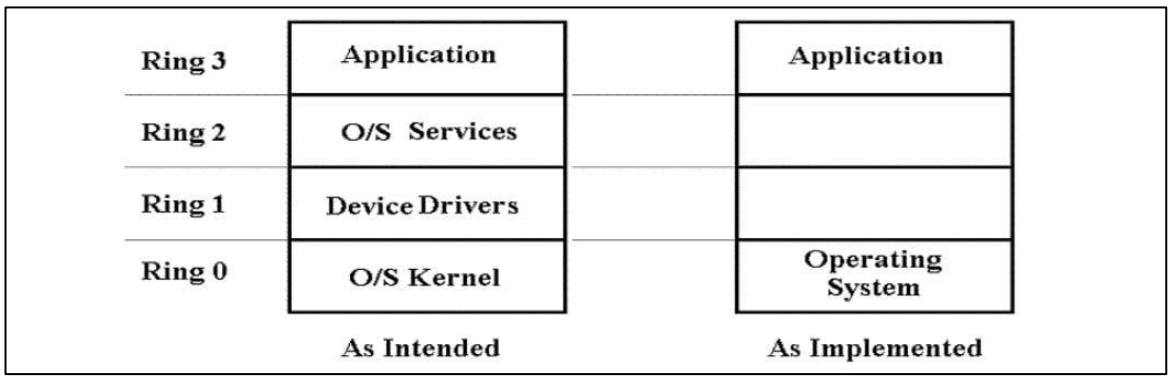
4. בנוסף לכך, תהליך כזה יכול לקבל שירות ברמת המערכת בצורה מוגבלת, על ידי ביצוע syscall שיקרא ל-system service, שיעשה עבורו שירות מסוים כגון Create File. יש שירותי מערכת שאפילו מתייחסים שונה לקריאה ממעגל ריצה 3, לפי PreviousMode שמועבר לשירות על ידי מערכת ההפעלה ומצהיר אם הקורא רץ ב-usermode (3) או ב-kernelmode (0):




### מילה קצרה על מעגלים 1/2:

בתיאוריה קיימים גם מעגלי אבטחה 1/2, שנועדו במקור ל-device drivers או תוכנות הקשורות לתקשורת של חומרה עם מערכת ההפעלה והיו צריכים יותר שליטה וחופש ממעגל ריצה 3, אך עדיין לא היו צריכים חופש ברמת מערכת ההפעלה.

בפועל לא משתמשים במעגלים האלה וכל device driver לרוב רץ במעגל 0 במקום:




זה כמובן מראה בעייה משמעותית ברוב מערכות ההפעלה - לתוכנה אין שום שליטה וחופש או שיש לה שליטה על הכל וחופש מוחלט.

**מעגל אבטחה 0 - בלתי מוגבל (לכאורה):**

עכשיו הגענו למעגל האבטחה "הכי גבוה" ברשימה - kernel mode. כפי שציינו כבר כל דבר שצריך יותר הרשאות מתהליך רגיל, כגון device drivers, מערכת ההפעלה עצמה וכל רכיב נוסף, רץ בצורה הבלתי מוגבלת הזו.

כל מה שרץ במעגל אבטחה 0 הוא non paged, כלומר, כל תוכנה קרנלית נמצאת על זכרון פיזי בכל רגע נתון בזמן ריצת המערכת, ואי אפשר לעשות לה paging (להחליף תוכן שלה בתוכן אחר). לכן, מרחב הכתובות הקרנלי הוא מרחב כתובות אמיתיות ומשותף לכל תוכנה קרנלית שרצה, כך שכל תוכנה מסוימת תוכל להשפיע על כל האחרות, כולל מערכת ההפעלה עצמה.

לתוכנות הקרנליות יש גישה לכל ה-system services בצורה ישירה על ידי exported functions מהקרנל שניתנים לכל תוכנה קרנלית, ומן הסתם שבצורות שונות כל תוכנה קרנלית תוכל להשפיע ולשנות את צורת הפעילות של כל תהליך שרץ במעגל אבטחה נמוך יותר. לכל תוכנה קרנלית יש גם גישה ישירה למשאבי המערכת, ויכולה לדרוש כמה שהיא רוצה ובאיזה צורה שהיא רוצה, אפילו לדרוש זכרון פיזי לכמה משתנים שישארו במרחב הכתובות של מערכת ההפעלה.

| Start | End | Size | Description |
|---|---|---|---|
| FFFF0800`00000000 | FFFFF67F`FFFFFFFF | 238TB | Unused System Space |
| FFFFF680`00000000 | FFFFF6FF`FFFFFFFF | 512GB | PTE Space |
| FFFFF700`00000000 | FFFFF77F`FFFFFFFF | 512GB | HyperSpace |

| | | | |
|---|---|---|---|
| FFFFF780`00000000 | FFFFF780`00000FFF | 4K | Shared System Page |
| FFFFF780`00001000 | FFFFF7FF`FFFFFFFF | 512GB-4K | System Cache Working Set |
| FFFFF800`00000000 | FFFFF87F`FFFFFFFF | 512GB | Initial Loader Mappings |
| FFFFF880`00000000 | FFFFF89F`FFFFFFFF | 128GB | Sys PTEs |
| FFFFF8a0`00000000 | FFFFF8bF`FFFFFFFF | 128GB | Paged Pool Area |
| FFFFF900`00000000 | FFFFF97F`FFFFFFFF | 512GB | Session Space |
| FFFFF980`00000000 | FFFFFa70`FFFFFFFF | 1TB | Dynamic Kernel VA Space |
| FFFFFa80`00000000 | *nt!MmNonPagedPoolStart-1 | 6TB Max | PFN Database |
| *nt!MmNonPagedPoolStart | *nt!MmNonPagedPoolEnd | 512GB Max | Non-Paged Pool |
| FFFFFFFF`FFc00000 | FFFFFFFF`FFFFFFFF | 4MB | HAL and Loader |

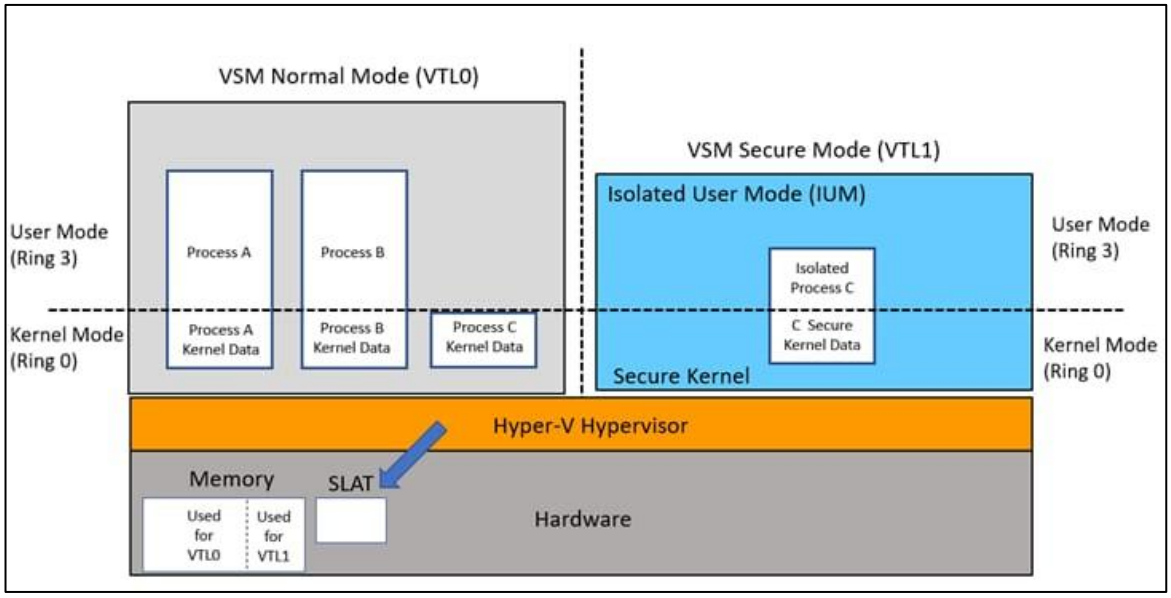
מפה אף אחד מהמעגלים לא נחשב רשמית למעגל אבטחה נוסף, אך עדיין נוספו בגלל ההגדרה המקבילה שלהם למעגל אבטחה נוסף - איזור שבו תהליך מסוים יכול לרוץ עם הרשאות מסוימות שנותנות לו בידוד וחופש כנגד כל מה שפועל במעגלי האבטחה מתחתיו (או בעצם מעליו במספר). הזכרון של האיזור הזה מבודד משאר הזכרון של מעגלי האבטחה הנמוכים יותר, אך יכול לגשת לזכרון של התהליכים שהם less privileged.

**מעגל אבטחה מינוס 1 - ?:**

מעגל האבטחה הזה הוא האחרון שבדרך כלל אנשים יודעים לתאר, מהסיבה שהתהליך המרכזי שרץ במעגל זה הוא ה-hypervisor.

ה-hypervisor הוא תהליך שנועד במקור לנהל מערכות הפעלה שונות שרצות על אותה מערכת פיזית, או בעצם מכונות וירטואליות. לתוכנה ניתנות הרשאות גבוהות יותר מהסיבה שהיא צריכה לנהל מערכות הרצות תחת אותה מערכת, ולמפות דברים כמו page tables של המכונות השונות ל-page tables של המערכת הפיזית.

בגלל צרכים אלו, hypervisor רץ כאילו במעגל אבטחה משל עצמו, בו אף תהליך שרץ במעגלים 0-3 יכול להשפיע עליו באיזשהי צורה:




ניתן לראות כאן קיום של הפרדה מכיוון שונה, תהליכים יכולים להיות תוכנות רגילות ב-usermode, ויכולים להיות תוכנות ב-kernelmode, אך יש הפרדה גם בין vtl0 (מערכת רגילה) לבין vtl1 (secure mod) בעזרת ה-hypervisor.

באופן כללי VTL0 הוא הרמה שבה מערכת ההפעלה הרגילה פועלת כאשר יש hypervisor פעיל. ברמה זו, למערכת ההפעלה יש גישה מוגבלת למשאבי החומרה מכיוון שהיא רצה כאורחת של ה-hypervisor.

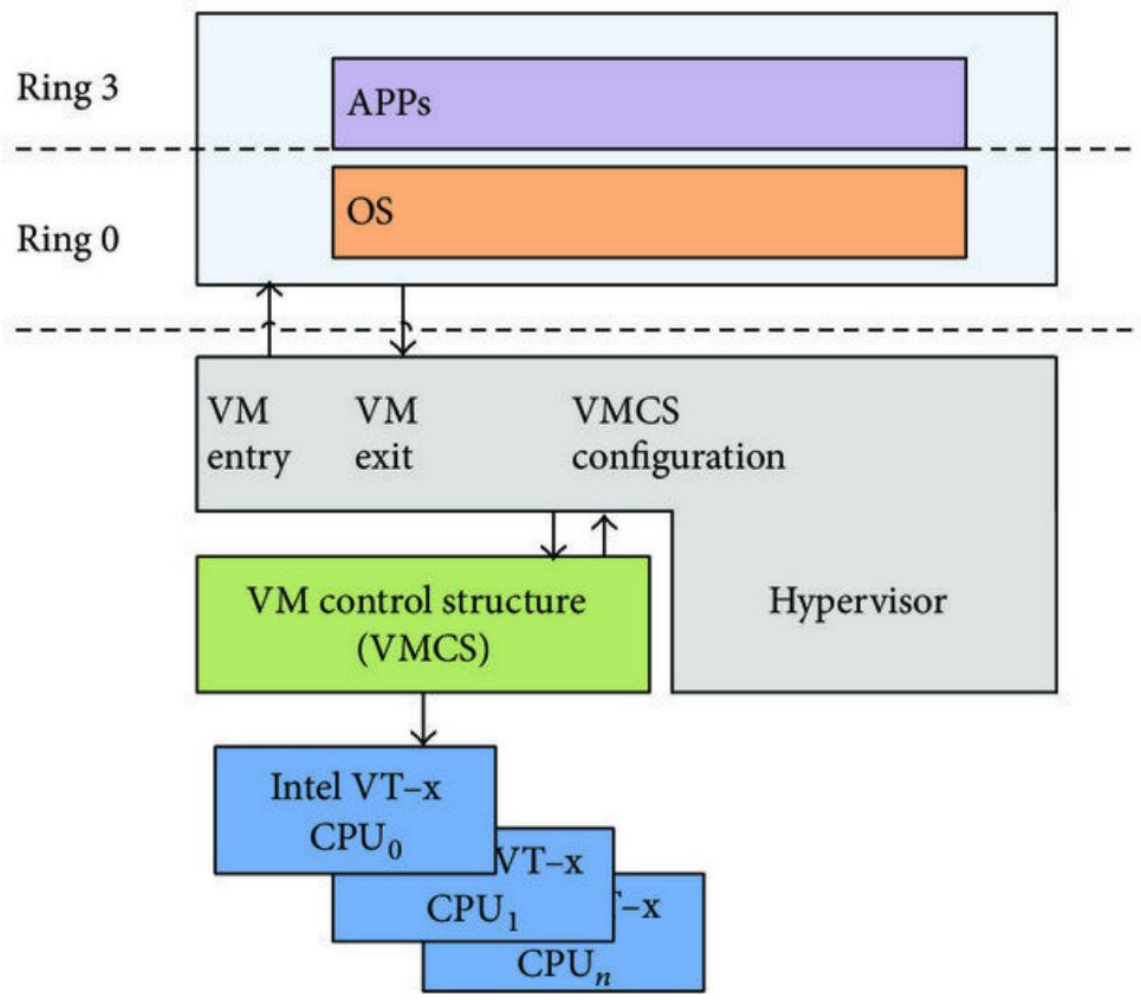
VTL1 היא רמה בעלת הרשאות גבוהות יותר שמשמשת את ה-hypervisor לניהול וירטואליזציה ואבטחה. ברמה זו, ה-hypervisor מבצע פעולות קריטיות כמו הפרדת משאבים ובידוד קוד זדוני.

ההבדל העיקרי הוא שרמת VTL1 שולטת בניהול המשאבים והאבטחה, בעוד ש-VTL0 פועלת עם גישה מוגבלת תחת הפיקוח של ה-hypervisor.

לסיכום, כאשר יש hypervisor שקיים על המערכת מתווסף לנו מעגל נוסף - מינוס אחד. במעגל הזה פועל ה-hypervisor ששולט על המיפוי של כתובות וירטואליות לכתובות פיזיות, בין אם עבור המערכת הרגילה ב-

page tables או עבור המערכות הוירטואליות דרך extended page table entries (מנגנון שעוזר למפות כתובות פיזיות של המכונה הוירטואלית לכתובות פיזיות של המכונה האמיתית). ה-hypervisor הוא בעצם הישות היחידה שרצה באיזור המבודד הזה של מעגל מינוס אחד, אבל הוא נותן אפשרות להריץ תוכנות במעגל אבטחה 0-3, שיהיו מוגנות משינויים של שאר המערכת, כאילו היו מופרדות גם בעוד מעגל אבטחה (vtl1). כמובן שגם ניהול שאר המשאבים, כגון syscalls/interrupts עוברים דרך ה-hypervisor.

כדי לנהל את משאבי מערכת ההפעלה המרכזית ומערכות ההפעלה הוירטואליות, כגון שמירת רגיסטרים, הרשאות ריצה, מידע על המעבד וכדומה, ה-hypervisor מנהל מבנה נתונים בשם VMCS. ה-hypervisor, שגם מנהל את תהליך ה-context switches, משתמש במבנה הזה כדי לשמור על מצב החומרה, תוך כדי החלפה בין הרצת קוד מתוך מכונה וירטואלית או תחת vtl0/vtl1:

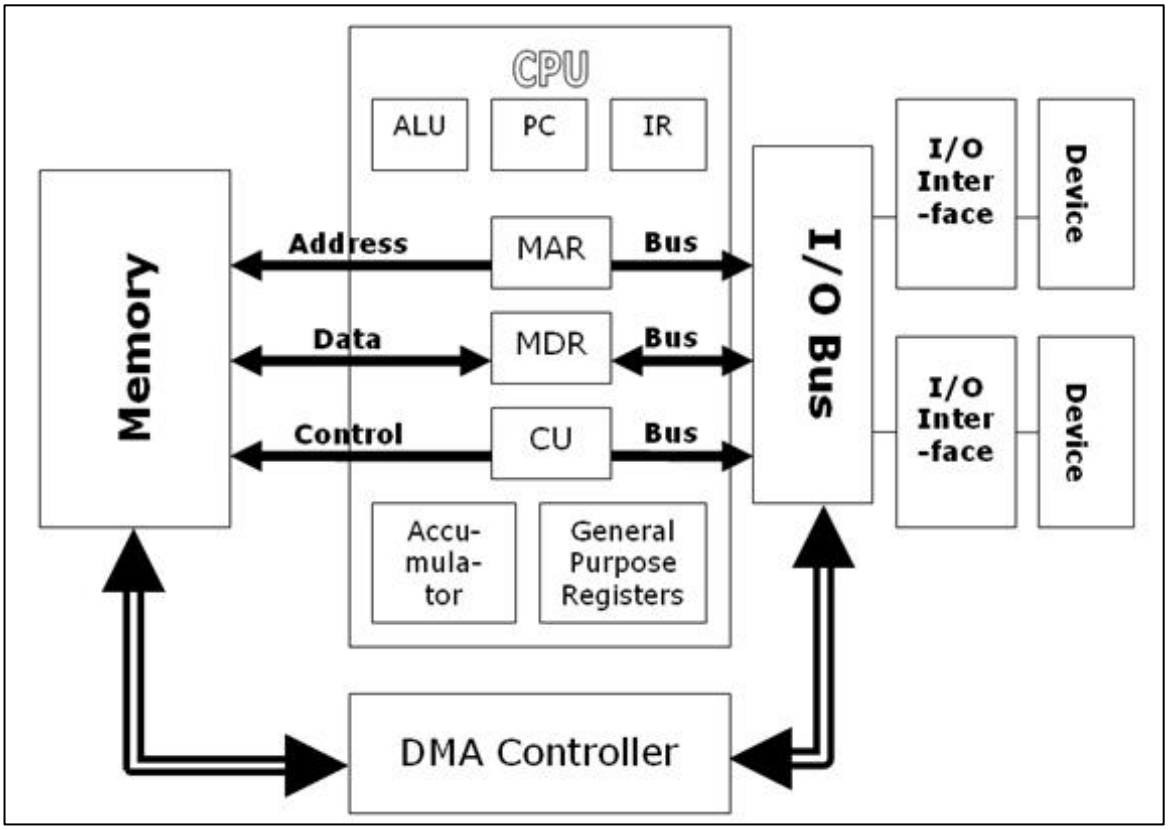



[מקור: https://www.researchgate.net/figure/Relationship-between-VM-and-hypervisor_fig1_323914683]

הדבר האחרון שכדאי לציין על hypervisors הוא השליטה שלהם על החומרה שמחוברת למערכת, בעזרת טכנולוגיה שנקראת IOMMU. DMA זו טכניקה פופולארית מאוד במערכות הפעלה מודרניות, שחוסכות

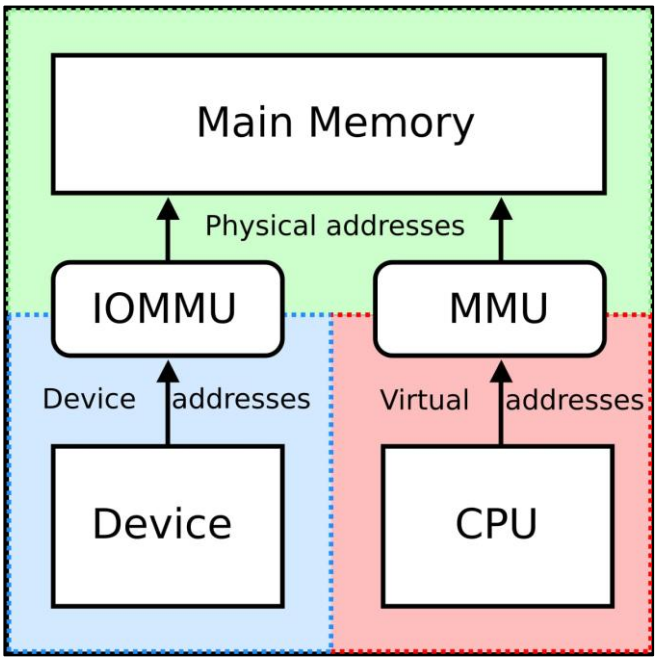
מהמעבד עבודה כשהתקן חומרה מסוים מחובר למערכת, בכך שההתקנים יכולים לגשת ישירות לזכרון המערכת בלי התערבות המעבד. יש פה יתרון משמעותי של מהירות ויעילות המערכת, אך ניתן לשים לב לפער המשמעותי באבטחת המערכת.

יש אפשרות שבה תנתן גישה להתקן זדוני לזכרון המערכת, או מה שנקרא DMA injection:




IOMMU הוא בעצם רכיב חומרה שאחראי לנהל את הכתיבה/קריאה על זכרון במערכת, שנעשה על ידי התקני חומרה. בכך הוא מספק שכבה נוספת של שליטה על רכיבי חומרה.

תהליך הפעילות שלו דומה ל-MMU הרגיל במערכת ההפעלה שפועל בעזרת ה-page tables, הרכיב הזה ממפה איזורי זכרון פנויים ספציפיים עבור אותם התקנים, ונותן להם לגשת רק לחלקים שהוקצו עבורם, כך שמתקפות כמו DMA injection ימנעו.




תחום ה-hypervisors מלא בעוד הרבה מידע ודוגמאות למנגנונים שונים שמומשו ב-hypervisor type 1/2, לכן אקשר למטה מקורות נוספים שאפשר להעזר בהם למי שרוצה ללמוד יותר על התחום.

**מעגל אבטחה מינוס 2 - ??:**

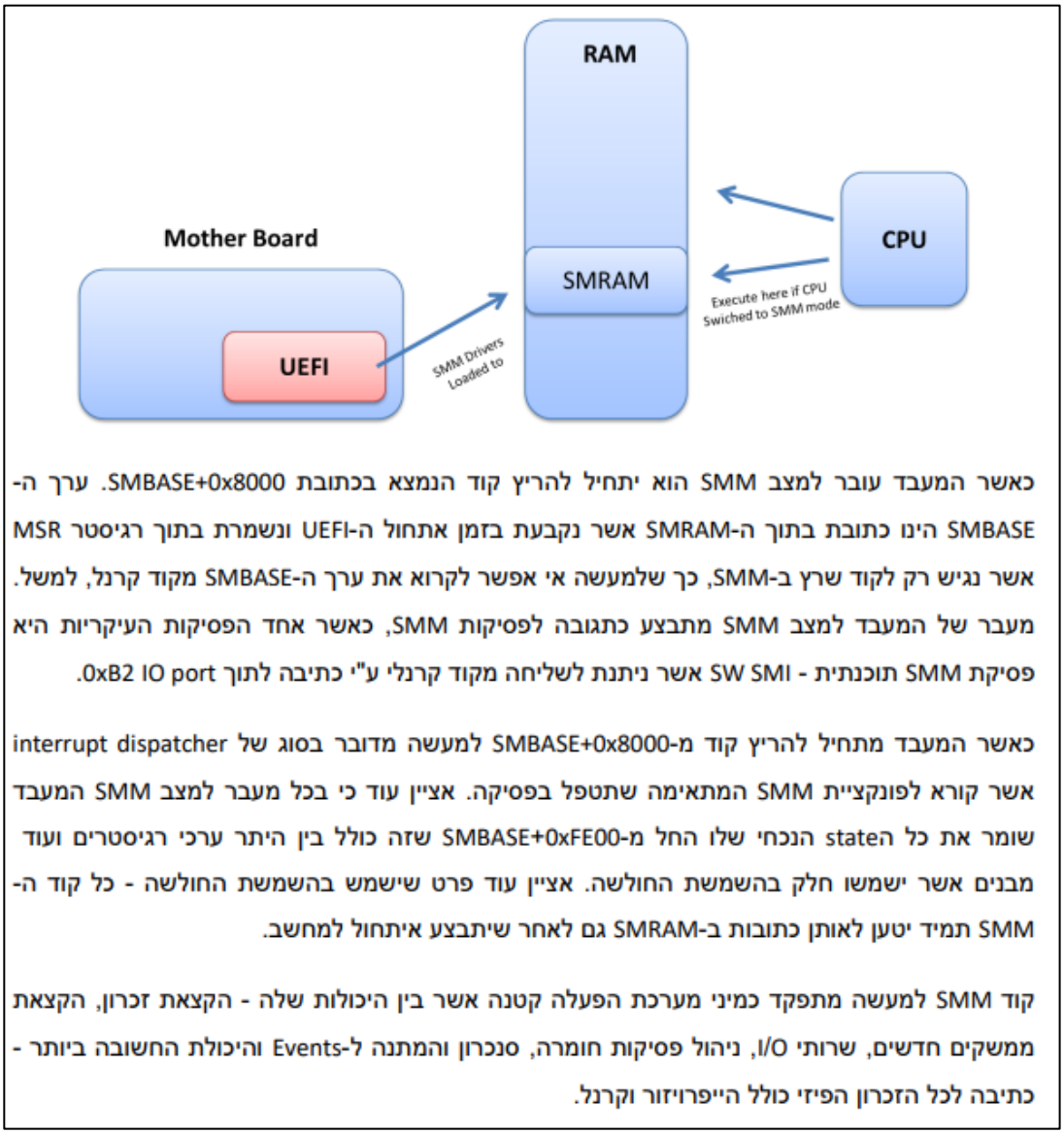
כדי להסביר את נושא ה-smm אעזר במאמר ממש מעניין של אביחי מינדר שאקשר למי שרוצה לקרוא עוד. SMM או system management mode, הוא מצב בו המעבד יכול לרוץ וירוץ במקרה שבו קורה אירוע מערכת שמצריך תגובת smm. במצב זה כל המערכת, כולל מערכת ההפעלה, מפסיקים לרוץ, ומערכת ה-smm שמאוכסנת בדרך כלל ב-firmware מתחילה לרוץ.

ריצת קוד smm מוטרג רק באירועים ספציפיים מאוד במערכת, כגון אירועי power management או hardware control שנועדו לשימוש רק על ידי ה-BIOS/UEFI.

כמו כן, לפי העיקרון שחזרתי עליו כל המאמר, הייחוד של מעגל ה-SMM היא שיש לו גישה לקריאה וכתיבה על כל זכרון פיזי קיים במערכת, כולל זכרון vtl0 וזכרון vtl1, כך שהוא עוקף את כל שאר המערכת ואת מעגל האבטחה שמתחתיו (hypervisor). זה חלק משמעותי מאוד, כי בכל מערכות ההפעלה המודרניות ה-

hypervisor נחשב לישות עליונה, וכל האבטחה המודרנית של המערכת, כגון hvci, שמבוססת על היכולות העליונות של מעגל מינוס 1, נשברות ברגע במקרה ורכיב smm קובע את זה.

הדרך האמיתית להריץ קוד שקיים ב-smm הוא בעזרת system management interrupt, שמריץ את הקוד במרחב כתובות נפרד בזכרון של המערכת- SMRAM. ה-firmware של מערכת ההפעלה מגביל את שאר המערכת מהאפשרות לכתוב/לקרוא מהזיכרון הפיזי הזה, וכך ממומשת ההגנה על האיזור שיוצרת את ההפרדה:




בפועל, כדי להטריג SMI כדי להכנס לריצת SMM יש מגוון דרכים:

1. ללוח האם יש אפשרות לשלוח signals מסוימים למעבד. מנגנון זה משומש ברגע שלוח האם או רכיב חומרה המחובר אליו צריך להודיע על אירוע מסוים, כגון לחיצה על כפתור או צורך מסוים של רכיב חומרה,

וגם עבור תקשורת מהמעבד אל לוח האם בנוגע לדברים כמו מתי לחסוך בכוח או כמה מהר לתת למעבד לרוץ.

התקשורת הזו מבוצעת דרך processor pins מסוגים שונים, שמאפשרים למעבד לתקשר עם רכיבי חומרה אחרים, כגון לוח האם, בנוגע למעבר מידע, ביקורת על המערכת ועוד. כל pin הוא בפועל חתיכת מתכת שמחברת בין 2 רכיבים או יותר, וה-processor pin שאחראי לטפל ב-SMIs הוא #SMI. ספציפית ה-signal שעובר דרך #SMI יכול לקרות בצורה עצמאית, כלומר לא תלוי ב-signals אחרים, או בקיצור stateless.

הינה טבלה של חלקם:

| Pin | Description |
|---|---|
| **CLK** | Clock input. Provides fundamental timing for the processor. |
| **CLKMUL** | Clock Multiplier pin (Bus Frequency).<br>Usually 2.0x or 3.0x, though it can be 2.5x or 4.0x, depending on the chip. |
| **D0 - D31** | Data pins. |
| **D/C#** | Data/Code. Primary bus cycle definition pin. |
| **DP0 - DP3** | Data Parity pins. |
| **EADS#** | External Address Strobe. |
| **FERR#** | Floating Point Error. |
| **FLUSH#** | Cache Flush. |
| **HITM#** | Hit to a Modified line. |
| **HLDA** | Hold Acknowledge. |
| **HOLD** | Bus Hold request. |
| **IGNNE#** | Ignore Numeric Error. |
| **INTR** | Maskable Interrupt. |
| **INV** | Invalidate. |
| **INVAL** | Invalidate. |
| **KEN#** | Cache Enable. |
| **KEY** | Key pin. Non-functional pin to prevent incorrect CPU insertion. |
| **LOCK#** | Bus Lock. |
| **M/IO#** | Memory/Input-Output. Primary bus cycle definition pin. |
| **MP#** | Math-coprocessor Present. When pulled low, the processor enters a powered-down tristate mode, allowing the math-coprocessor to take control. |
| **NMI** | Non-Maskable Interrupt. |
| **PCD** | Page Cache Disable. |
| **PCHK#** | Parity Status. |
| **PLOCK#** | Pseudo-Lock. |
| **PWT** | Page Write-Through. |
| **RDY#** | Non-burst Ready. |
| **RESET** | Reset. |
| **RPLSET0** | ?. |
| **RPLSET1** | ?. |
| **RPLVAL#** | ?. |
| **SMADS#** | System Management Interrupt Address Strobe. |

| | |
|---|---|
| **SMI#** | System Management Interrupt. Allows processor to enter system management mode. |
| **SMIACT#** | System Management Interrupt Active. Indicates that processor is in system management mode. |
| **SRESET** | Soft Reset. |
| **STPCLK#** | Stop Clock. |
| **SUSP#** | Suspend. Same as STPCLK#. |
| **SUSPA#** | ?. |
| **TCK** | Test Clock. |
| **TDI** | Test Data Input. |
| **TDO** | Test Data Output. |
| **TEST** | Test pin. |
| **TMS** | Test Mode Select. |
| **UP#** | Upgrade Present. When pulled low, the processor enters a powered-down tristate mode, allowing an upgrade processor to take control. |

למי שמעוניין ללמוד עוד על processor pins הינה קישור לאתר שמתאר הרבה מהמידע החשוב למטה.

2. דרכים נוספות לבצע SMI צריכות להתבצע במודע לחומרת המערכת, לדוגמא כתיבה לכתובת בזכרון שה-firmware הגדיר למעבד, או שליחת I/O ל-hardware port ששמור עבור הגיון של לוח האם (בדרך כלל פורט 0xB2).

כפי שתיארתי בחלק מהמאמרים הקודמים, hardware ports הם דרכים ניהוליות ל-I/O devices של חומרה חיצונית המחוברת למערכת, לדוגמא מקלדת או עכבר. לניהול יש API ברור, שנותן לכתוב/לקרוא כמות מסוימת של בתים מפורט מסוים, ודרכו נשלחות הודעות לגבי אירועי חומרה שונים, כגון SMI שבפועל נועד לכל מיני סוגים של hardware malfunctions.

13.8.2.1 **APM_CNT—Advanced Power Management Control Port Register**

| | | | |
|---|---|---|---|
| I/O Address: | B2h | Attribute: | R/W |
| Default Value: | 00h | Size: | 8-bit |
| Lockable: | No | Usage: | Legacy Only |
| Power Well: | Core | | |

| Bit | Description |
|---|---|
| 7:0 | Used to pass an APM command between the OS and the SMI handler. Writes to this port not only store data in the APMC register, but also generates an SMI# when the APMC_EN bit is set. |

עכשיו, איך בפועל נשלח המידע ל-SMI handler ב-SMM?

1. מתבצע SMI בצורה מסוימת, אחת מהאפשרויות והסיבות שציינתי במאמר.

2. במערכת קיים רגיסטר בשם SMBASE register. ערך ברירת המחדל שלו הוא 0x30000 והוא מצביע לבסיס ה-SMM בזכרון הפיזי של המערכת. ברירת המחדל היא שה-entry point של ה-SMM נמצא 0x8000 בתים אחרי ה-SMBASE, כלומר בדרך כלל הפקודה הראשונה שתרוץ היא ב-0x38000.

3. נקודת הכניסה הגנרית בדרך כלל מבצעת בדיקות טיפול כלליות, כגון לבדוק למה נעשה SMI. הקוד עושה זאת בעזרת ה-SMI status registers שמביאים ל-SMM גישה לפרמטרים שונים בנוגע לסיבת הכניסה.

4. ל-SMM יש "טבלה" של פעולות שאותם הוא מריץ עבור כל סוג אפשרי של סיבת ביצוע ה-SMI. כמובן שבזמן ריצת הפעולה כל interrupt רגיל שיכול להפריע למערכת להתרכז ב-SMM מנוטרל, כדי שהפעולה הנ"ל תסיים את הריצה שלה ללא הפרעה.

5. בנוסף לפורט 0xB2 שבדרך כלל משמש להטריג את ה-SMI ולספק status code (בדומה ל-IOCTL code שקורה ב-drivers רגילים), יש בדרך כלל פורט נוסף שעוזר לספק מידע נוסף על ה-SMI. בארכיטקטורה של x86 זה בדרך כלל 0xB3 וכמובן שצריך לכתוב אליו לפני ה-SMI (רק כתיבה ל-0xB2 מטריגה SMI, אך שהיא מוטרגת לא יהיה אפשר לכתוב כבר ל-0xB3. אפשר לחשוב על זה כמו שבניהול קלט/פלט מ-driver רגיל יש את הקוד של סיבת הכניסה (כתיבה, קריאה, open handle, close handle וכדומה), ואז פרט מידע נוסף שמספק מידע נוסף על הכניסה (ioctl code).

6. הקלט שכל פעולת SMM יודעת לקבל מוגדר לפי מבנה נתונים בסיסי של הארכיטקטורה, שמספק מידע כמו data size ו-data payload שרלוונטיים לפעולה. כתובת המבנה הזה קבועה בזכרון המערכת או שמועברת באחד מה-SMI status registers, תלוי בארכיטקטורה.

   יש כמובן סיטואציות שמשתנות בין מקרה למקרה, לדוגמא סיטואציות שבהן אין שימוש במבנה פקטה להעברת מידע, ויש רק שימוש ב-0xB2 וב-0xB3.

   הגישה השנייה בדרך כלל לא משומשת במערכות מודרניות, בגלל הקושי שקיים להעביר כמות משמעותית של מידע שהפעולה כנראה צריכה.

7. אין דרך ברירת מחדל להחזרת מידע מה-SMM כ-return value, אז הנושא מאוד תלוי מימוש. ברגע סיום הפונקציה, הקוד של ה-SMM מבצע RSM, שהיא פקודה מיוחדת שיכולה לרוץ רק ב-SMM, שנותנת למעבד אפשרות להריץ את שאר המערכת שוב ולצאת מה-SMM.

**הערת הכותב:** בגלל שהמאמר הופך להיות ארוך כבר, אז את הדוגמא הפרקטית לשימוש ב-smm אדגים במאמר אחר, אבל בינתיים למי שרוצה לקרוא וללמוד עצמאית יש פרויקט מעניין שמדגים את הקונספט שאקשר למטה.

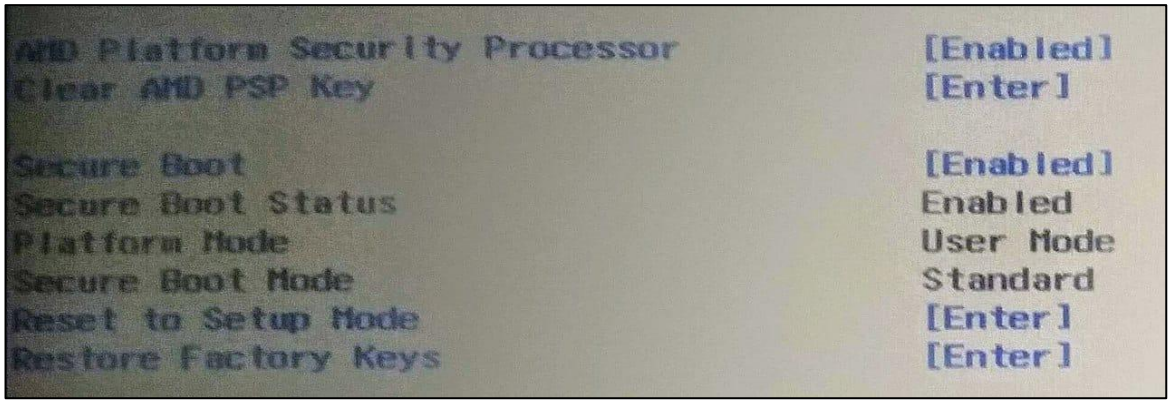
**עד מתי???**

מתחיל להיות מסובך, נכון?

לאט לאט זה מתחיל להשמע יותר ויותר לא מציאותי, אבל יש עוד מעגל אחד ששווה לדבר עליו.

כפי שניתן לראות בתמונה הראשונה יש עוד שלב אחרון-ME. ה-Intel management engine הוא כמו מחשב בפני עצמו, שרץ במעבד של כל מחשב בו קיים מעבד של אינטל.

המערכת הזאת, שרוב האנשים לא יודעים שקיימת בכלל, רצה כל הזמן שבו לוח האם מקבל כוח מהספק, גם כשהמחשב מכובה לגמרי. אז את האמת שלא רק ל-intel יש מערכת עליונה כזו, גם AMD מימשו מערכת דומה בשם Platform Security Processor:




אז בעצם איך המערכת נראית?

- המערכת רצה על יחידת עיבוד קטנה בתוך המעבד המרכזי, שמוקצית רק עבור ה-management engine.
- המערכת היא בעצם firmware נפרד ששאר המכונה לא מודעת אליו.
- מטעמי אבטחת מידע, קשה מאוד למצוא מידע על הפונקציונליות של המערכת, מכיוון והיא ברובה undocumented, ועברה obfuscation רב בעזרת האלגוריתם Huffman coding.
- האלגוריתם בקצרה מאוד הוא שיטה לדחיסת נתונים, שבה סמלים שמופיעים יותר מקבלים קודים קצרים, וסמלים שמופיעים פחות מקבלים קודים ארוכים.
- השוס במערכת זה שהטבלה שקובעת את כל ההמרות של האלגוריתם נמצאת אך ורק על החומרה, כך שגם אם נרצה להשתמש במידע ה-firmware עבור פענוח לא נוכל לעשות זאת.

***"..there is no generic way to turn it off."***

ה-firmware נמצא על חלק מסוים מה-bios flash, וחלק מהבידוד שלו משאר המערכת מתבטא גם בתחום הרשתות.

ל-ME יש גישה ישירה ל-ethernet controller, עם כתובות MAC ו-IP משלו, כשאפילו חלק מסוים של תעבורת ה-ethernet תכנס ל-ME לפני שהיא נכנסת למערכת ההפעלה.

***"The Intel ME is still on, even when your computer is off."***

כפי שניתן להבין, אין באמת דרך לרוץ ידנית כחלק ממעגל אבטחה מינוס שלוש, אך גם אם לא מכיוון של יצירת נוזקה שפועלת במעגל הזה, עדיין קיימת בעיית אבטחת מידע חמורה שברורה מאוד לעין:

***"The idea of the NSA putting hardware in every computer sounds absurd, until you realize it actually happened."***

הבעייה המרכזית קורית כשיש מערכת כל-יכולה, שלא ניתן לכבות ולא ניתן להבין אותה עד הסוף, אך מהצד השני נתונה עדיין לקוד בעייתי ואפילו חולשות קריטיות:

## Security vulnerabilities [ edit ]

Several weaknesses have been found in the ME. On May 1, 2017, Intel confirmed a Remote Elevation of Privilege bug (SA-00075) in its Management Technology.[35] Every Intel platform with provisioned Intel Standard Manageability, Active Management Technology, or Small Business Technology, from Nehalem in 2008 to Kaby Lake in 2017 has a remotely exploitable security hole in the ME.[36][37] Several ways to disable the ME without authorization that could allow ME's functions to be sabotaged have been found.[38][39][12] Additional major security flaws in the ME affecting a very large number of computers incorporating ME, Trusted Execution Engine (TXE), and Server Platform Services (SPS) firmware, from Skylake in 2015 to Coffee Lake in 2017, were confirmed by Intel on 20 November 2017 (SA-00086).[40][41] Unlike SA-00075, this bug is even present if AMT is absent, not provisioned or if the ME was "disabled" by any of the known unofficial methods.[42] In July 2018, another set of vulnerabilities was disclosed (SA-00112).[43] In September 2018, yet another vulnerability was published (SA-00125).[44]

זה רק סיכום של חלק מהבעיות שהיו במוצר, לכן אקשר את העמוד בתחתית המאמר למי שרוצה להתעמק במקרים ספציפיים. אך את המאמר אני אסיים בזה, שגם אם המערכת כל-כך קריטית ומרכזית במכונה, עדיין יכולה להיות פשלה קטנה שברורה מאליו לרוב שיכולה לשנות את גורל המחשבים של הרבה מאיתנו.

## סיכום

לסיכום, עברתי במאמר הזה על כל נושאי ההרשאות של חלקי קוד שונים במערכת המודרנית, צורות הריצה השונות והקונספט שכל מעגל בעצם עוטף את המעגל הקודם, עם שליטה מוחלטת עליו כשלמעגל הקודם אין שום שליחה חוזרת.

אחרי זה נכנסתי לפרטים על כל מעגל, איך הוא ממומש ועובד, ולבסוף כל הבעיות שיכולות לעלות משימוש בו לצורך זדוני / מבעיות אבטחה שקיימות במימוש שלו.

## על המחבר

בן 18, חוקר חולשות ב-**Cyberillium**. מתעניין מאוד בתחומי הפיתוח ומחקר בסביבת **Lowlevel**, מערכות הפעלה ואבטחת מידע. מעוניין מאוד לפתח את הידע שלי וללמוד עוד כדי להתפתח בתחום. בין הפרויקטים המרכזיים שלי עבדתי על **rootkit** למערכת ההפעלה **Windows 10** כדי להחביא תהליכים, קבצים ותעבורת רשת. כמוכן שגם פיתחתי מערכת להגנה מנוזקות קרנליות כמו שלי ואחרות. בנוסף לכך פיתחתי וחקרתי דרייברים ומבני נתונים פנימיים רבים.

ניתן לראות את הפרויקטים האלו ואחרים בעמוד הגיטהאב שלי:

https://github.com/shaygitub

ניתן ליצור איתי קשר דרך האימייל שלי:

shaygilat@gmail.com

או דרך עמוד הלינקדאין שלי:

https://www.linkedin.com/in/shay-gilat-67b727281

## ביבליוגרפיה

**מאמרים בנושא הייפרוויזורים:**


https://www.digitalwhisper.co.il/files/Zines/0x7C/DW124-1-NativeHyperVisoer.pdf

https://www.digitalwhisper.co.il/files/Zines/0x7D/DW125-1-NativeHyperVisoer-Part2.pdf

https://www.digitalwhisper.co.il/files/Zines/0x72/DW114-4-HyperV-Security.pdf

https://www.digitalwhisper.co.il/files/Zines/0x09/DW9-5-Virtualization.pdf

https://rayanfam.com/topics/hypervisor-from-scratch-part-1/


**מאמרים בנושא SMM:**


https://www.digitalwhisper.co.il/files/Zines/0x78/DW120-1-SMMVulnIntro.pdf

https://www.pchardwarelinks.com/486pin.htm

https://electronics.stackexchange.com/questions/295239/where-can-i-find-the-definition-of-smi-sci-pins-on-a-cpu

https://github.com/Cr4sh/SmmBackdoorNg


**נושא ה-ME:**


https://en.m.wikipedia.org/wiki/Intel_Management_Engine


קישורים למאמרים הקודמים שלי בנושאי Windows Internals שיכולים לעזור להבין את שאר המאמרים שלי יותר טוב ולהכנס בצורה יותר חלקה לתחום (יש יותר מדי לכן אקשר מעכשיו את עמוד הגיטהאב שמתעד את כולם):


https://github.com/shaygitub/Articles