Digital Whisper
Technologic Papers

פורצים את הדיסק: מתקפות UEFI ב-20₪

מאת רן ורשוור

הקדמה

השאלה שהניעה את המחקר הזה הייתה פשוטה: עד כמה באמת אפשר לסמוך על מערכת כאשר תוקף מחזיק בגישה פיזית לחומרה? ובאופן יותר ספציפי איך מנגנוני הצפנת הדיסק של מיקרוסופט כגון Bitlocker (להלן: ביטלוקר) עובדים מתחת למכסה המנוע, ולהבין איך אפשר לעקוף אותם עם כמה כלים זולים מאלי אקספרס.

המאמר הזה נוצר בהשראה רבה מהחומרים המצוינים של OpenSecurityTraining2 שהציתו את הסקרנות שלי ושלחו אותי למסע הזה.

ביטלוקר 101: מפתחות, TPM וקריפטוגרפיה

לפני שאנחנו מנסים לפצח את ההצפנה, כדאי להבין כיצד היא בכלל אמורה לעבוד. ככל שמבינים יותר טוב את העיצוב האלגנטי שלה כך קל יותר לראות היכן מתחילים להופיע הסדקים (במחסור) במנגנוני האבטחה שלה.

ביטלוקר הוא ניסיון של מיקרוסופט להתמודד עם אתגר די רציני: כיצד מחשב יכול לפתוח את הדיסק המוצפן בצורה אוטומטית בפני משתמש מורשה אבל לסרב לכל מי שאינו מורשה?

הפתרון בנוי על תהליך הנקרא Trusted Boot, שרשרת מדידות קריפטוגרפיות המתחילה בקושחת המערכת (UEFI/BIOS) וממשיכה רכיב אחר רכיב תוך כדי תהליך האתחול של המחשב. כל שלב מודד את הבא אחריו באמצעות hash וכך נבנית חתימה ייחודית של מצב המערכת בעת ההגדרה הראשונית של ההצפנה.

כיצד עובדת שרשרת האמון

מרכיב מרכזי בתהליך הוא שבב הנקרא Trusted Platform Module, או בקיצור TPM. זהו שבב ייעודי שמחזיק בתוכו את המפתחות הקריפטוגרפיים, וישחרר אותם רק אם רצף המדידות שהוזכר קודם תואם לערכים שנקבעו בעת הפעלת ביטלוקר לראשונה. אם מתרחש שינוי משמעותי בשרשרת האתחול, ה-TPM יסרב לשחרר את המפתחות ולעיתים יגרום למערכת להיכנס למצב שחזור.

תהליך זה יוצר את מה שמכונה ה-Root of Trust, "שורש האמון", שממנו הכל נבנה ונמדד.

אך כמו כל שרשרת, היא חזקה רק כחוליה החלשה שבה. ה-TPM אמנם מחזיק את המדידות, אך הוא לא זה שמבצע אותן בפועל, הוא מסתמך לחלוטין על הערכים שהקושחה שולחת אליו. אם תוקף שינה את הקושחה, היא יכולה לשלוח ל-TPM ערכי מדידה/hashes מזויפים שייראו "תקינים". מכיוון של-TPM אין יכולת לאמת את מקור המדידה, הוא יקבל אותם כמות שהם וישחרר את המפתחות למרות שהמערכת כבר נפרצה.

מי מודד את המודד?

אז כמו שהבנו, לביטלוקר יש בעיה בסיסית בהיגיון הפעולה שלה: הקושחה אמורה למדוד את כל מה שבא אחריה - אבל אף אחד לא מודד את הקושחה עצמה ברגע שהיא עולה. ה-TPM פשוט מקבל את ערכי המדידה שהקושחה שולחת לו.

כדי להבין כיצד ה-TPM מקבל את ההחלטה האם לשחרר או לחסום מפתחות, צריך להכיר קבוצת רגיסטרים קריטית לתהליך: Platform Configuration Registers, או בקיצור PCRs. ה-PCRs הם אוסף של רגיסטרים, שכל אחד מהם משמש מעין יומן קריפטוגרפי שאליו מוזרמות המדידות במהלך האתחול. כל שלב מודד את הבא אחריו ומכניס את הערך לתוך ה-PCR הרלוונטי, וכך נבנית החתימה הקריפטוגרפית של מצב המערכת.

הבעיה היא שה-TPM אינו מבצע את המדידות בעצמו ואינו יודע מי יצר אותן כלומר אם תוקף שינה את הקושחה כך שתשקר לגבי ערכי ה-PCR, ה-TPM לא יטיל שום ספק בערכים שנמסרו לו וישחרר את מפתחות הביטלוקר כל עוד הוא מקבל את הערכים שהוא מצפה להם מבלי שום ידיעה שהמערכת כבר שבויה בידי התוקף.

זו הבעיה הקונספטואלית של שורש אמון סטטי (CRTM): הרכיב הראשון שמריץ קוד חייב להיחשב מהימן, ואין לשאר המערכת דרך לאמת זאת. וברגע שהקושחה עצמה נמצאת בשליטת התוקף, כל שרשרת האמון שנבנית מעליה הופכת לחסרת משמעות.

להלן מבנה ה-PCRs:

```
Initial startup FW at CPU reset vector
PCR[0] ← CRTM, UEFI Firmware, PEI/DXE [BIOS]
- UEFI Boot and Runtime Services, Embedded EFI OROMs
- SMI Handlers, Static ACPI Tables
PCR[1] ← SMBIOS, ACPI Tables, Platform Configuration Data
PCR[2] ← EFI Drivers from Expansion Cards [Option ROMs]
PCR[3] ← [Option ROM Data and Configuration]
PCR[4] ← UEFI OS Loader, UEFI Applications [MBR]
PCR[5] ← EFI Variables, GUID Partition Table [MBR Partition Table]
PCR[6] ← State Transitions and Wake Events
PCR[7] ← UEFI Secure Boot keys (PK/KEK) and variables (dbx..)
PCR[8] ← TPM Aware OS specific hashes [NTFS Boot Sector]
PCR[9] ← TPM Aware OS specific hashes [NTFS Boot Block]
PCR[10] ← [Boot Manager]
PCR[11] ← BitLocker Access Control
```

חשוב להבין שה-PCRs לא פשוט מוחקים ומעדכנים את הערך הקודם, אלא עובדים במנגנון הנקרא Hash Extension. כל מדידה חדשה מתווספת לערך הקיים באמצעות הנוסחה:

```
(PCR[new] = SHA256(PCR[old] || measurement)
```

משמעות הדבר היא שכל שינוי ברצף האתחול ישנה לחלוטין את הערך הסופי של ה-PCR.

החוליה החלשה: זיכרון ה-SPI Flash

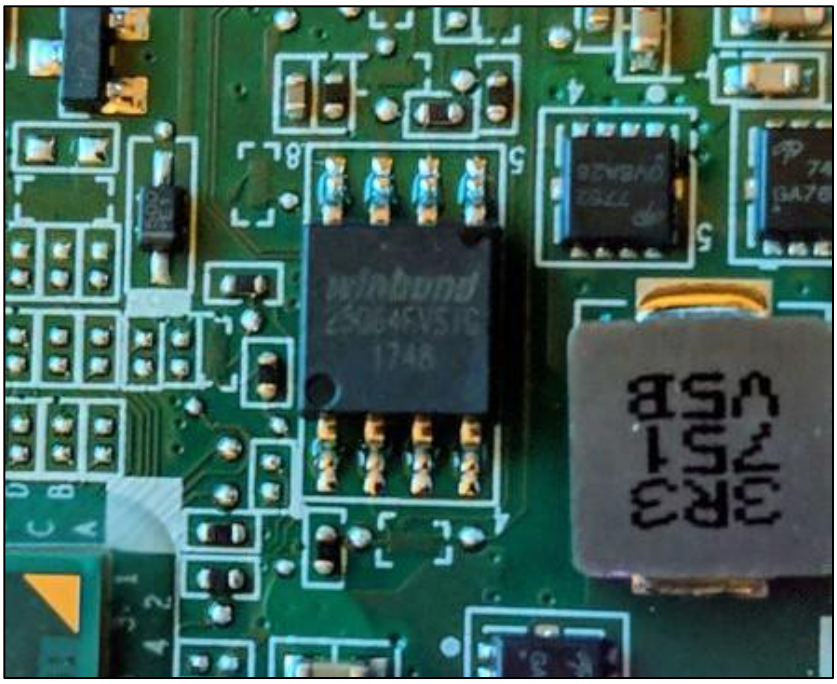
כעת, מכשהבנו ששרשרת האמון נשענת כולה על קוד הקושחה שנקרא ראשון בעת הפעלת המערכת, מגיע הרגע לשאול את השאלה המתבקשת: "איפה בכלל מאוחסן הקוד הזה ומדוע כל כך קל לשבש אותו?"

על כל לוח מודפס של מכשיר, ספציפית לוח אם הקושחה מאוחסת על שבב קטן חיצוני מסוג SPI Flash. זהו התקן זיכרון פשוט יחסית שמחובר לשאר הרכיבים על גבי הלוח בקווי תקשורת ייעודים (SPI Bus) ונגיש לחלוטין ברמת החומרה.

וזה בדיוק העניין, למרות שמדובר באחד הרכיבים הכי רגישים ומשמעותיים מבחינה אבטחתית הגישה אליו כמעט ולא מוגבלת - כלומר כל מה שנדרש לעשות הוא לפתוח כמה ברגים ולמצוא את השבב על הלוח.

תרגיל! קחו לוח אם ישן או כל לוח מודפס (PCB) שנמצא בבית וחפשו שבב קטן בעל שמונה רגליים, הקלידו את הטקסט שנמצא עליו בגוגל ותוכלו לגלות שרוב היצרים מפרסמים datasheet מקיף כולל פרטים על מבנה הזיכרון והפרוטוקולים הפנימיים.

להלן דוגמא:



כאן נכנס לתמונה מתכנת הפלאש הזול שהזמנו מאלי אקספרס: כלי שמאפשר לנו לקרוא ולכתוב ישירות את הקושחה המאוחסנת על שבב הפלאש. אמנם המפתחות של ביטלוקר מאוחסנים על הדיסק ומוצפנים באמצעות ה-TPM, קושחה שעברה שינוי ונכתבה מחדש יכולה לבצע פעולות כגון לשלוח ל-TPM מדידות מזויפות שנראות תקינות, או להשתיל קוד שירוץ בשלבי האתחול המוקדמים וילכוד את מפתחות ההצפנה ברגע שהם משתחררים לזיכרון.

האירוניה הגדולה היא שהחוזקה הכי משמעותית של ביטלוקר - ההתבססות העמוקה על שרשרת האתחול היא גם נקודת התורפה הגדולה ביותר שלו. כל רכיב בשרשרת האמון הופך למשטח תקיפה פוטנציאלי עבור תוקף שמחזיק גישה פיזית לחומרת המערכת.

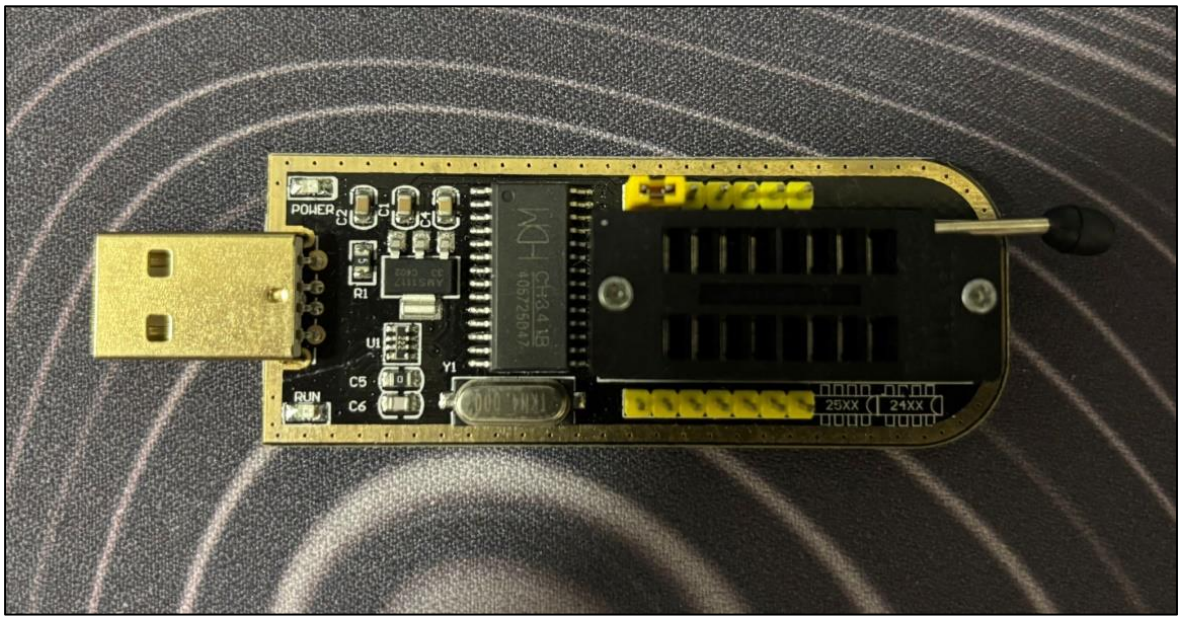
מתקפת ה-20₪: הכלים

היופי במתקפה הזאת היא הנגישות שלה. אין צורך בציוד יקר או במעבדה מקצועית, כל מה שצריך אפשר להזמין מאלי אקספרס ב-20₪ או פחות ויגיע תוך שבוע בשקית פלסטית מפוקפקת.

מתכנת ה-CH341A

הלוח הקטן הזה הוא הכלי המרכזי שלנו, הוא מיועד לתכנות שבבי פלאש ו-EEPROM שונים ותופס מצוין לצורך שלנו לקריאה וכתיבה מזיכרון הפלאש. ה-CH341A מדבר בפרוטוקול SPI ומזוהה כמכשיר USB סטנדרטי במחשב שלנו.

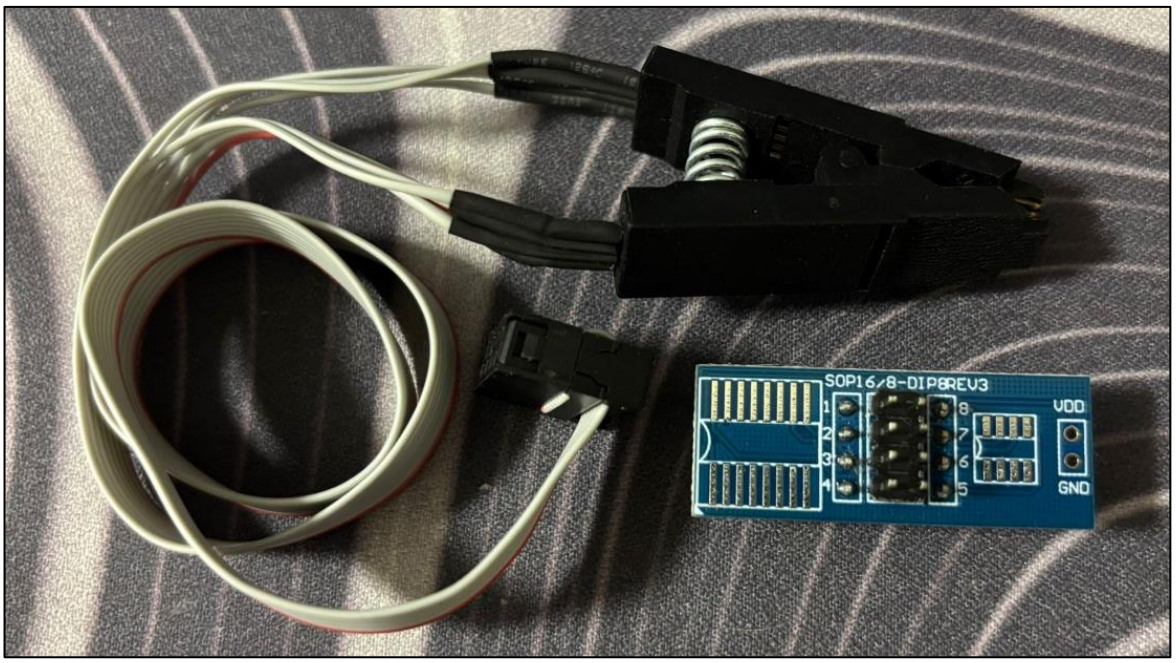
כך הוא נראה:



אזהרה: ה-CH341A עלול לפלוט V5 במקום V3.3 הנדרש לרוב שבבי הפלאש, מה שעלול לגרום נזק פיזי.
יש לוודא הגדרת מתח נכונה לפני החיבור.

קליפס SOIC8

בדרך כלל יגיע כחלק מהחבילה יחד עם ה-CH341A. הקליפס מתחבר ישירות לרגלי השבב ללא צורך בהלחמה או פירוק השבב מהלוח מה שהופך את המתקפה ללא הרסנית ומהירה במיוחד. הינה הוא:



שליפת הקושחה מהחומרה

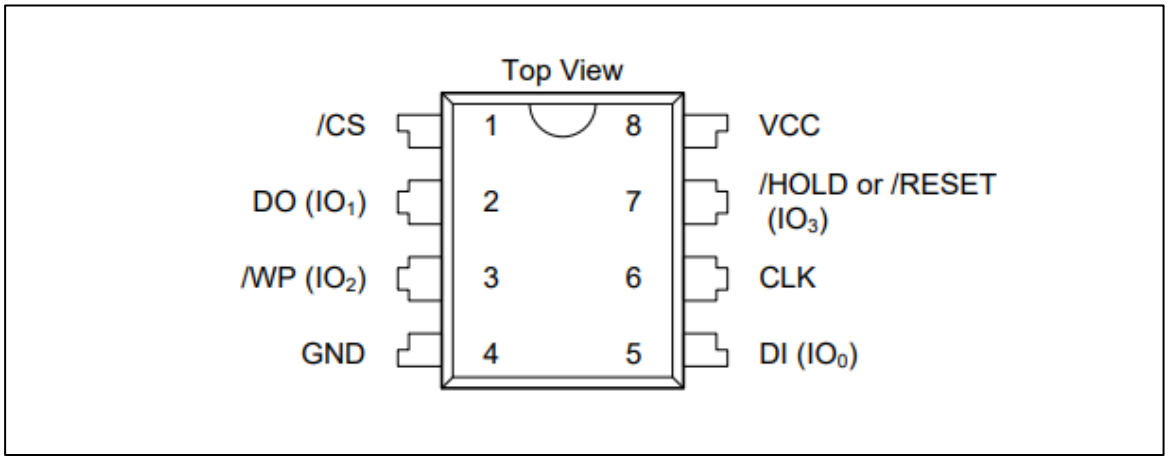
אחרי שניגשים ללוח עלינו למצוא את שבב הפלאש, שבדרך כלל יגיע בתצורת SOIC8 או במילים אחרות שבב קטן עם שמונה רגליים המולחם ישירות על הלוח המודפס.

לעיתים השבב יכול להסתתר מתחת לכבל שטוח או מאחורי מגן מתכת ובמקרים כאלו נצטרך להזיז את כבלים בעדינות או לשחרר עוד כמה ברגים עד שנקבל גישה מספקת לשבב. על לוחות מסוימים השבב יכול להימצא בצד הפנימי של הלוח כלומר אם מסתכלים כדוגמא על מחשב נייד, נצטרך להוציא את כל הלוח מהמארז ולהפוך אותו לצידו השני.

חשוב מאוד לחבר את הקליפס בכיוון הנכון, הכבל שמסמל את פין 1 בדרך כלל צבוע באדום ועל השבב עצמו נראה נקודה קטנה בפינה. אם אין על השבב שום סימון או דרך לזהות את פין 1 נוכל להציץ ב-datasheet ולקבל משם תמונה יותר ברורה.

חיבור לא נכון בין הקליפס לשבב יגרום לקריאה שגויה במקרה הטוב, או נזק פיזי לרכיבים במקרה הפחות טוב.

להלן מבנה פין-אאוט סטנדרטי של שבב פלאש:



Winbond 3V 128M-BIT SERIAL FLASH MEMORY WITH DUAL/QUAD SPI & QPI

הכנת סביבת עבודה

לאחר חיבור מתכנת הפלאש השלב הבא הוא שימוש בכלי flashrom - כלי אופן סורס שמיועד לקריאה, כתיבה, גיבוי ושחזור של קושחה התומך במגוון סוגי שבבים ופרוטוקולים סריאלים. הכלי מזהה את סוג השבב ומאפשר לנו לחלץ עותק גולמי בפורמט בינארי שאותו לאחר מכן נוכל לשכתב ולכתוב חזרה לשבב, או לבצע בו הנדסה לאחור והשוואות (diffing) בין עותקים וגרסאות שונות של הקושחה.

ראשית נוודא שמערכת ההפעלה מצליחה לזהות את המתכנת והשבב, ונקבל חזרה פלט הכולל את סוג השבב, נפחו ופרוטוקול התקשורת שלו:

```
flashrom -p ch341a_spi
flashrom v1.2 on Linux
Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
Found Winbond flash chip "W25Q64.V" (8192 kB, SPI) on ch341a_spi.
```

קריאת השבב עשויה לקחת מספר דקות בהתאם לגודל הזיכרון(בדר"כ MB8) וליציבות החיבור. שלב קריטי בתהליך הוא לוודא שהדאמפ קושחה שחילצנו הינו תקין משום שקריאות יכולות להיכשל אם הקליפס לא ישב בצורה מושלמת ולכן מומלץ לבצע קריאה כפולה ולהשוות בין ה-checksums של שני הבינארים שקיבלנו:

```
sudo flashrom -p ch341a_spi -r firmware_dump2.bin
sha256sum firmware_dump.bin firmware_dump2.bin
```

אם ה-checksums תואמים נוכל להיות בטוחים שהדאמפ תקין, ובמידה ויש חוסר התאמה נשחק קצת עם הקליפס או ננסה לחבר אותו מחדש לגמרי עם דגש על מיקומי הרגליים בהתאם לפין-אאוט.

לאחר אימות תקינות הדאמפ, חשוב ליצור גיבוי של הקובץ המקורי. הגיבוי מאפשר לשחזר את הקושחה המקורית במקרה של ניסויים, טעויות כתיבה או עדכונים מסוכנים, ומבטיח שהמערכת תישאר מתפקדת גם לאחר פעולות מסובכות או ניסויים עם קוד הקושחה.

ניתוח זיכרון הקושחה

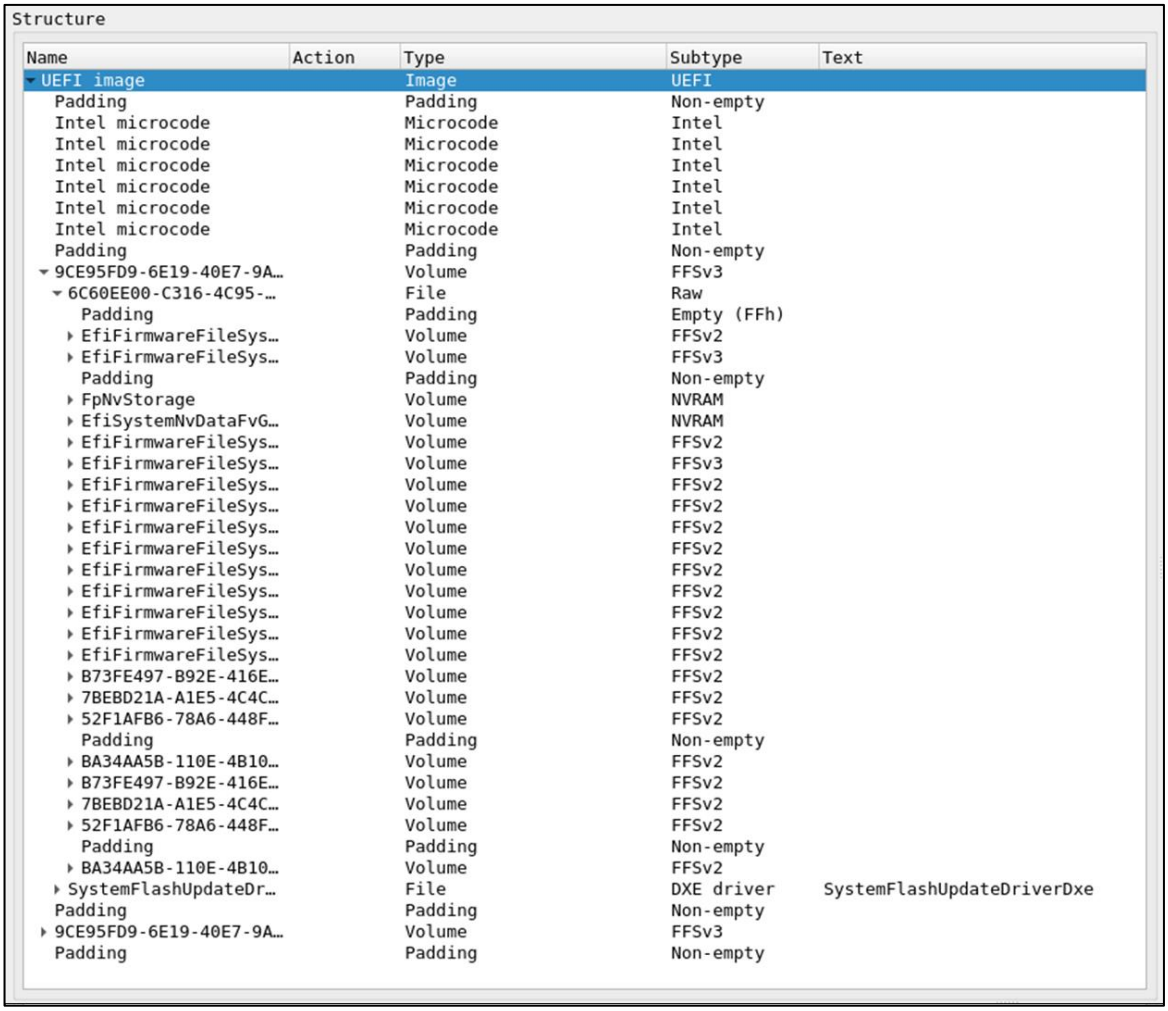
עם חילוץ הקושחה מתגלה בפנינו מבנה המכיל למעשה את כל מה שהמערכת צריכה כדי לרוץ, אפשר לחשוב על זה כמו קובץ ZIP שמכיל עוד תתי ZIP-ים בתוכו שלכל אחד יש מבנה פנימי שונה. בשלב זה נכנס לפעולה כלי נהדר נוסף בשם UEFITool, עוד כלי אופן סורס המיועד לניתוח, פירוק וניווט בקושחת UEFI. הכלי למעשה יודע לפרש את ה-Flash Descriptor של אינטל ולהציג את כל חלוקת אזורי הזיכרון בשבב בצורה גרפית מה שמאפשר חקירה מדויקת של כל רכיבי הקושחה.

במבט על נוכל לראות את אזורי הפלאש העיקריים:

- Flash Descriptor - מגדיר את חלוקת האזורים בשבב
- BIOS Region - מכיל את קוד ה-UEFI ואת ה-NVRAM
- ME Region - קושחת Intel Management Engine
- GbE Region - תצורת בקר הרשת
- PDR Region - נתוני פלטפורמה נוספים

אזור ה-BIOS הוא היעד המרכזי שלנו. בתוכו נמצאים:

- מודולי UEFI בינאריים (דרייברי DXE, מודולי PEI)
- מחסן משתני NVRAM - המקום שבו שמורים משתני EFI
- הגדרות אתחול ותצורת מערכת
- ולעיתים גם מידע הקשור להתנהגות Bitlocker ולמדיניות האבטחה שלו



חילוץ משתני ה-NVRAM

כעת הגיע הזמן להתמקד באזור ה-NVRAM שמכיל בפועל את הגדרות המערכת. למזלינו UEFITool מאפשר לנו לעשות זאת בצורה דיי פשוטה, מאתרים וליום NVRAM ובוחרים Extract body כדי לשמור לקובץ בינארי את החלק הרלוונטי שאותו נוכל לפתוח בעוד חלון של UEFITool:

Structure

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
EfiSystemNvDataFvGuid		Volume	NVRAM	
VSS2 store		VSS2 store		
EfiCustomModeEnabl…		VSS entry	Auth	CustomMode
Invalid		VSS entry	Invalid	
EfiVendorKeysNvGuid		VSS entry	Auth	VendorKeysNv
Invalid		VSS entry	Invalid	
4D1EDE05-38C7-4A6A…		VSS entry	Auth	FirmwareFeatures
4D1EDE05-38C7-4A6A…		VSS entry	Auth	FirmwareFeaturesMask
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 1
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 2
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 3
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 4
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 5
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 6
Invalid		VSS entry	Invalid	
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 7
IScsiConfigGuid		VSS entry	Auth	InitialAttemptOrder
EfiIScsiInitiatorN…		VSS entry	Auth	Attempt 8
Invalid		VSS entry	Invalid	
EfiGlobalVariableG…		VSS entry	Auth	Boot0000
Invalid		VSS entry	Invalid	
EfiGlobalVariableG…		VSS entry	Auth	PlatformLang
EfiGlobalVariableG…		VSS entry	Auth	Lang
EdkiiVarErrorFlagG…		VSS entry	Auth	VarErrorFlag
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
EfiGlobalVariableG…		VSS entry	Auth	Key0000
EfiGlobalVariableG…		VSS entry	Auth	Key0001
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	
Invalid		VSS entry	Invalid	

לאחר פתיחת הקובץ ניתן לראות את המשתנים האמיתיים שהקושחה משתמשת בהם בזמן האתחול ובתקשורת עם מערכת ההפעלה, וספציפית את מבנה ה-2VSS שבו מאוחסנים משתני ה-EFI.

בין המשתנים הנפוצים שנראה:

- CustomMode - מצב Secure Boot (רגיל או Custom)
- VendorKeysNv - מפתחות יצרן
- FirmwareFeatures / FirmwareFeaturesMask - יכולות שהקושחה מצהירה לעומת יכולות זמינות
- 0000Boot וערכים דומים - ערכי אתחול
- 0001Key0000, Key - מפתחות Secure Boot
- PlatformLang, Lang - הגדרות שפת מערכת
- IScsiConfigGuid - הגדרות iSCSI לאתחול רשת
- VarErrorFlag - סימון שגיאות פנימי

נוכל לראות מספר ערכים המסומנים כ-Invalid, אלו משתנים שנמחקו לוגית אך עדיין קיימים בדאמפ מאשר והקושחה לא לא מוחקת נתונים ישנים מיד אלא רק מסמנת אותם כלא בשימוש.

למטרת התקיפה עצמה אין צורך אמיתי בניתוח עמוק של מבנה הקושחה ותוכן ה-NVRAM. אנחנו לא צריכים להבין כל מודול DXE או מה מייחס כל משתנה במערכת, אבל כן חשוב לקבל תמונה כללית על איך הקושחה מאורגנת, איפה מאוחסנים המשתנים, ואיך רצף האתחול נראה מבפנים.

הגנה על זיכרון הפלאש

בפלטפורמות אינטל מודרניות קיימים מנגנוני הגנה המבוססים על ההנחיות המוגדרות בתקן NIST SP 800-147 BIOS Protection Guidelines. מנגנונים אלו מיושמים באמצעות רגיסטרים ייעודיים בתוך מערך השבבים, שתפקידם להגן על אזורי הפלאש הקריטיים ולמנוע כל ניסיון בלתי מורשה לשנות את הקושחה.

BIOS_CNTL - מנגנוני ההגנה המרכזיים

רגיסטר BIOS_CNTL שולט בשאלה מי רשאי לגעת בקושחה ומתי. הוא כולל שלושה מנגנוני אבטחה עיקריים:

- **SMM_BWP - הגנת כתיבה באמצעות SMM:** מנגנון שמחייב כניסה למצב System Management Mode על מנת לכתוב לקושחה.
- **BLE - מנגנון נעילת BIOS:** מפעיל מלכודת (SMI) בכל פעם שתהליך מנסה לאפשר כתיבה לקושחה. המטרה היא לאפשר לקושחה עצמה או ל-SMM ליירט ניסיונות כתיבה לא חוקיים ולמנוע פגיעה.

- **BIOSWE - מתג כתיבה לקושחה:** זהו הדגל שבפועל פותח או סוגר את הדלת לכתיבה. כאשר שאר המנגנונים נעולים, שינויו אינו מתאפשר וזהו מנגנון האכיפה האחרון שמטרתו להבטיח שהקושחה תישאר לקריאה בלבד.

13.1.32 BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	20h	Size:	8 bit
Lockable:	No	Power Well:	Core

Bit	Description
7:6	Reserved
5	**SMM BIOS Write Protect Disable (SMM_BWP)**— R/WLO. This bit set defines when the BIOS region can be written by the host. 0 = BIOS region SMM protection is disabled. The BIOS Region is writable regardless if processors are in SMM or not. (Set this field to 0 for legacy behavior) 1 = BIOS region SMM protection is enabled. The BIOS Region is not writable unless all processors are in SMM.
4	**Top Swap Status (TSS)** — RO. This bit provides a read-only path to view the state of the Top Swap bit that is at offset 3414h, bit 0.
3:2	**SPI Read Configuration (SRC)** — R/W. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3 – Prefetch Enable Bit 2 – Cache Disable Settings are summarized below: **Bits 3:2** — **Description** 00b — **No prefetching, but caching enabled.** 64B demand reads load the read buffer cache with "valid" data, allowing repeated code fetches to the same line to complete quickly 01b — **No prefetching and no caching.** One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10b — **Prefetching and Caching enabled.** This mode is used for long sequences of short reads to consecutive addresses (i.e., shadowing). 11b — **Reserved. This is an invalid configuration**, caching must be enabled when prefetching is enabled.
1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

[מקור: Intel® 6 Series Chipset and Intel® C200 Series Chipset Datasheet]

PR0-PR4 - מנגנוני הגנת טווחים

רגיסטרי ה-Protected Range מגדירים חמישה אזורים מהפלאש מוגנים מכתיבות חיצוניות:

- **Write Protection Enable:** כאשר דגל זה פעיל החומרה חוסמת כל ניסיון כתיבה או מחיקה לאזור שהוגדר על ידי הבסיס והגבול. זוהי אכיפה ברמת החומרה, שאינה תלויה בתוכנה.
- **Read Protection Enable:** כאשר דגל זה פעיל, הקריאות לאזור מוגן נחסמות גם הן על ידי החומרה.
- **Protected Range Base / Limit:** מגדירים את הגבולות המדויקים של האזור המוגן כחלק מהטווח שמוגן על ידי WPE/RPE.

21.1.13 PR0—Protected Range 0 Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 74h Attribute: R/W
Default Value: 00000000h Size: 32 bits

Note: This register can not be written when the FLOCKDN bit is set to 1.

Bit	Description
31	**Write Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:29	Reserved
28:16	**Protected Range Limit —** R/W. This field corresponds to FLA address bits 24:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	**Read Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that read directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:13	Reserved
12:0	**Protected Range Base** — R/W. This field corresponds to FLA address bits 24:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

[מקור: Intel® 6 Series Chipset and Intel® C200 Series Chipset Datasheet]

המציאות? ההגנות לרוב מושבתות מהמפעל

מחקרים הראו שבעבר ואצל חלק גדול מהיצרנים מהמערכות מגיעות מהמפעל עם מנגנוני האבטחה האלו מושבתים לחלוטין, כל הרגיסטרים מאופסים ל-0x00 והקושחה ניתנת לכתיבה אפילו ישירות דרך מערכת ההפעלה ללא צורך במתכנת פלאש או גישה פיזית מיוחדת לחומרה.

גם במערכות שבהן ההגנות מוגדרות כראוי, חשוב להבין שמנגנונים אלו מגנים בעיקר מפני מתקפות מבוססות תוכנה ולא מספקים הגנה מלאה מול תוקף בעל גישה פיזית למערכת:

```
OS     : Windows 7 6.1.7600 AMD64
Chipset:
  VID:        8086
  DID:        0100
  Name:       Sandy Bridge (SNB)
  Long Name: Sandy Bridge CPU / Cougar Point PCH

[+] loaded common.bios_wp
[+] imported chipsec.modules.common.bios_wp
[x][ ================================================================
[x][ Module: BIOS Region Write Protection
[x][ ================================================================
BIOS Control (BDF 0:31:0 + 0xDC) = 0x08
[05]    SMM_BWP = 0 (SMM BIOS Write Protection)
[04]    TSS     = 0 (Top Swap Status)
[01]    BLE     = 0 (BIOS Lock Enable)
[00]    BIOSWE  = 0 (BIOS Write Enable)

[-] FAILED: BIOS region write protection is disabled
[*] BIOS Region: Base = 0x00180000, Limit = 0x003FFFFF

SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0
[-] FAILED: None of the SPI protected ranges write-protect BIOS region
```

כאשר מתחברים פיזית לשבב הפלאש באמצעות מתכנת כמו ה-CH341A כל מנגנוני ההגנה שלו פשוט נעקפים לחלוטין. המתכנת מדבר ישירות עם החומרה ולכן רגיסטרים כמו BIOS_CNTL ו-PRx לא משחקים שום תפקיד בהגנה על הקושחה.

<u>מה מנגנוני האבטחה כן מונעים:</u>

- נוזקות/פוגענים המנסים לשכתב את ה-BIOS מתוך מרחב המשתמש/קרנל במערכת ההפעלה.
- עדכוני קושחה לא מורשים שמגיעים מאותו המרחב גם כן.
- נסיונות השרדה של פוגענים לתוך הקושחה(rootkits).

<u>ומה הם לא מונעים בשום צורה:</u>

- תקיפות המתבססות על חיבור מתכנת פלאש לשבב.
- גישה לחומרה העוקפת לחלוטין את בקרי ה-PCH/SPI
- תקיפות המתבססות על תזמון נעילות רגיסטרים באתחול מוקדם

חתימות, הנחיות והמצב בפועל

באופן אינטואיטיבי אפשר היה לצפות שקבצי הקושחה חתומים קריפטוגרפים ומאומתים לפני הרצה, כך שכל שינוי קטן היה גורם למערכת לסרב לעלות. אבל בפועל זה לא המצב ברוב המערכות הנפוצות.

תקן ה-NIST SP 800-147 BIOS Protection Guidelines שהזכרנו לפני כן קובע שקיימת *המלצה* לאמת עדכוני קושחה בצורה קריפטוגרפית אך חשוב לשים לב שמדובר רק בהנחייה ולא בדרישה מחייבת. כתוצאה מכך יצרנים רבים מיישמים אימות רק כחלק מתהליך עדכון רשמי ולא כחלק מתהליך האתחול עצמו:

2. BIOS Update Authentication

2-A There shall be a Root of Trust for Update (RTU) that contains a signature verification algorithm and a key store that includes the public key needed to verify the signature on the BIOS update image.

2-B The key store and the signature verification algorithm shall be stored in a protected fashion on the computer system and shall be modifiable only using an authenticated update mechanism or a secure local update mechanism as outlined in Section 3.1.2.

2-C The key store in the RTU shall include the public key for verifying the signature on a BIOS update image or include the hash [FIPS 180-3] of the public key for verifying the signature on a BIOS update image that includes the public key. In the latter case, the update mechanism shall ensure that the hash of the public key provided with the BIOS update image appears in the key store before using the provided public key to verify the signature on the BIOS update image.

2-D BIOS images shall be signed in conformance with NIST SP 800-89, *Recommendation for Obtaining Assurances for Digital Signature Applications* [SP800-89], using an approved digital signature algorithm as specified in NIST FIPS 186-3, *Digital Signature Standard* [FIPS186-3], that provides at least 112 bits of security strength, in accordance with NIST SP 800-131A, *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths* [SP800-131A].

2-E The authenticated update mechanism shall ensure that the BIOS update image has been digitally signed and that the digital signature can be verified using one of the keys in the key store in the RTU before updating the BIOS.

[מקור: NIST BIOS Protection Guidelines]

בפועל, מנגנון Secure Boot מאמת רק רכיבים מסוימים בשרשרת האתחול כגרון מודולי UEFI נפרדים (כגון דרייברי DXE ואפליקציות UEFI), בוט לואדרים של מערכת ההפעלה, ולעיתים גם Option ROMs. לעומת זאת, הקושחה כמכלול אינה נחתמת או מאומתת, אזור משתני ה-NVRAM אינו מוגן קריפטוגרפית, שלבי אתחול מוקדמים כגון SEC ו-PEI פועלים לפני שמנגנוני האימות נכנסים לתוקף, והקוד שאחראי בעצמו על בדיקת החתימות אינו מאומת על ידי אף גורם חיצוני.

המשמעות היא ש-Secure Boot מגן היטב על רכיבים שמורצים לאחר טעינת הקושחה, ואינו מספק הגנה מפני כל מה שקשור לשינויים זדוניים בקושחה עצמה או ברכיבים הפועלים בשלבי האתחול המוקדם. זהו פער עקרוני המאפשר לתוקף בעל גישה פיזית להשתיל קוד שישפיע על כל שרשרת האתחול ועדיין יראה תקין בעיני המנגנונים הקריפטוגרפיים שאמורים לאמת את אבטחת המערכת.

הבנת ההקשר בין ה-PCR לביטלוקר

כדי שהמתקפה תעבוד הקושחה חייבת לדעת אילו ערכי PCR משפיעים על הגנת מפתחות הביטלוקר. המערכת אינה בודקת את כל ה-PCRs אחד אחרי השני אלה עובדת בצורה סיסטמטית המוגדרת מראש שנשמרת כחלק מה-metadata של ביטלוקר, נוכל לראות זאת גם בעצמינו:

```
PS C:\Windows\system32> manage-bde -protectors -get C:
BitLocker Drive Encryption: Configuration Tool version 10.0.19041
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

Volume C: []
All Key Protectors

    TPM:
      ID: {61C3AB9A-0638-4D02-AB28-0FCA4B9F01E2}
      PCR Validation Profile:
        0, 2, 4, 11
```

בדוגמא זו ניתן לראות שביטלוקר מתייחס לארבעה PCRs עיקריים ומבין כולם PCR[0] הוא היעד המרכזי של התקיפה. זהו המקום בו נמדדת הקושחה עצמה, הרכיב הראשון ברשרת האמון.

איך נראית המתקפה

בתרחיש הכי סביר, התוקף כבר ביצע את כל עבודת המחקר והפיצ'פוץ של הקושחה מראש, זיהוי הפונקציות הקריטיות ומשטחי ההזרקה. ברגע האמת, כל מה שנדרש הוא חלון קצר של גישה פיזית למערכת.

השתלשלות האירועים בפועל תיראה בצורה הבאה:

1. קבלת גישה פיזית וכתיבת הקושחה הזדונית

בעת קבלת גישה פיזית למחשב/מערכת של הקורבן לדוגמא בתרחיש "Evil Maid" התוקף יתחבר ישירות לשבב הפלאש באמצעות מתכנת חומרה. מאחר שהכתיבה מתבצעת מחוץ לשרשרת האתחול ומבלי לעבור דרך בקרים או מנגנוני ההרשאה של הקושחה, רגיסטרים כמו BIOS_CNTL או PRx אינם רלוונטיים, וכל ההגנות שתוכננו למנוע כתיבה מתוך מערכת ההפעלה פשוט אינן נכנסות למשחק.

2. אתחול המערכת והרצת הקושחה החדשה

באתחול הבא של המערכת הקוד הראשון שירוץ על המעבד הוא בעצם הקוד של הקושחה החדשה ששונתה. במהלך שלבי האתחול מתבצעות מדידות קריפטוגרפיות של רכיבים שונים, אך הלוגיקה האחראית על כך שונתה. כאשר הקושחה מגיעה לנקודה שבה היא אמורה לחשב hash של רכיב מסוים ולהרחיב אותו אל תוך ה-PCR הרלוונטי, היא אינה משתמשת ב-hash של הקוד שרץ בפועל אלא בערך מחושב מראש התואם למדידות המקוריות שבוצעו בעת הפעלת ביטלוקר וכך נוצר מצב שבו הקוד בפועל שונה, אך ה-TPM מקבל תמונה נקייה של המערכת.

3. שחרור המפתחות תחת הנחות שגויות

לאחר שהקושחה מסיימת את תהליכי המדידה ומדווחת ל-TPM ערכי PCR זהה לחלוטין לזה שנמדד בעת ההגדרה הראשונית לביטלוקר ה-TPM מניח שמצב הפלטפורמה לא השתנה. מאחר וכל תנאי האימות בוצעו מבחינתו הוא משחרר את מפתח ה-VMK המשתמש לפתיחת ההצפנה של הדיסק.

בשלב זה נוצר הפער הקריטי - מפתחות ההצפנה משוחררים לזכרון של מערכת שכבר שבויה בפני קושחה זדונית ללא שום סימן או מנגנון המעיד על כך. בידי התוקף האפשרות לקרוא את המפתחות ישירות מהזכרון ולשמור אותו לפענוח של הדיסק אופליין בכל עת או לחלץ לרשת קבצים רגישים מהמערכת שהרגע נפתחה בפניו.

סיכום ומסקנות

מה שהתחיל מסקרנות לגבי הצפנות דיסק ומתקפות פיזיות הפך לגילוי של פער מהותי בעיצוב האבטחה של הארכיטקטורה: הקוד שרץ ראשון ייחשב מהימן כברירת מחדל ובמרבית מהמערכות הצרכניות הוא חי בשבב שנגיש לחלוטין, וברגע שלתוקף יש גישה כזו כל מודל שרשרת האמון מתחיל להתפורר.

הגנה בשכבות

למרות זאת, קיימים מספר צעדים מעשיים שיכולים לצמצם משמעותית את משטח התקיפה:

- TPM + PIN - הפעלת PIN בעת הגדרת הביטלוקר מחייבת מהתוקף להשיג סיסמא אחרת שינוי הקושחה בלבד כבר אינו יעיל.
- בדיקת הגדרות המערכת - כפי שראינו, אחוז לא קטן מהמערכות מגיעות מהמפעל עם מנגנונים מושבתים או בתצורת ברירת מחדל שאינה מאובטחת, מומלץ לבדוק ולהגדיר ידנית את ההגנות.
- שימוש בחומרה מודרנית חתומה מראש - מעבדים מודרניים של אינטל ו-AMD תומכים במנגנונים שבהם מפתחות האימות צרובים פיזית במעבד עצמו (Hardware Root of Trust) מה שמצריך את הקושחה להיות חתומה ע"פ הנהלים שהיצרן מגדיר.
- אבטחה פיזית - גם אם לא עקבתם עד כאן והחלטתם שלא ליישם אף אחת מההמלצות, פשוט לא להשאיר את המכשירים ללא השגחה זה העקרון הכי חשוב עד כה. שתי דקות של גישה למכשיר מאפשר לתוקף עולם ומלואו מבחינת יכולות התקפיות.

הלקח הסופי הוא שלא מדובר רק בביטלוקר, כל אמצעי אבטחה מתבסס על הנחות לגבי מה אמין ומה לא, ההגנה האמיתית באה מהבנה של איפה ההנחות האלו יכולות להתנפץ.

עבור רוב המשתמשים ביטלוקר הנשען על TPM בלבד מספק הגנה בסיסית מפני רוב האיומים, גנב שחוטף לנו את המחשב בבית קפה כנראה לא יצליח לפרוץ את הצפנת הדיסק. אך עבור מטרות בעל ערך גבוהה או תרחישים של מתקפות ממוקדות, מודל האיומים משתנה לחלוטין ונדרשת חשיבה של מה אנחנו מגדירים כמאובטח באמת.

וזו אולי המסקנה הלא נוחה ביותר: **מדידות קריפטוגרפיות לא שוות הרבה, אם אף אחד לא מודד את מי שמבצע את המדידה**.

אם חידשתי לכם משהו, או שהנעתי אתכם לצאת למסע מחקר חומרה בעצמכם תרגישו חופשי להתחבר אלי או לעיין בבלוג (הריק למדי) שלי!

מקורות מידע

- https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-147.pdf
- https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/6-chipset-c200-chipset-datasheet.pdf
- https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/countermeasures
- https://www.pjrc.com/teensy/W25Q128FV.pdf
- https://web.archive.org/web/20160610025935/https://cansecwest.com/slides/2013/Evil%20Maid%20Just%20Got%20Angrier.pdf
- https://blog.elcomsoft.com/2021/01/understanding-bitlocker-tpm-protection/
- https://opensecuritytraining.info/IntroBIOS_files/Day1_00_Advanced%20x86%20-%20BIOS%20and%20SMM%20Internals%20-%20Motivation.pdf
- https://blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf