Digital Whisper
Technologic Papers

# איך לחטוף מיגרנה (CVE-2023-32369)

מאת יונתן בר אור

## הקדמה

מאמר זה מסכם מחקר שפורסם על ידי קבוצת Microsoft Defender ופורסם במאי 2023.

המחקר מתאר כיצד חיפוש שגרתי של קוד זדוני על גבי טלמטריות EDR, גרם לגילוי design issue ב-macOS שעוקף את מנגנון SIP. לאחר השמשת החולשה הצוות דיווח על הממצאים לחברת אפל, שסגרה את החולשה (כעת ידועה גם כ-CVE-2023-32369).

מטרת מאמר זה היא הנגשת המחקר בעברית ותיאור של מספר פרטים נוספים שהושמטו מפרסום המחקר לצורך נוחות קריאה.

## קצת מידע על SIP

בדומה ל-SE Linux על מערכות לינוקס, גם ב-macOS משתמש root אינו באמת כל יכול - מנגנון בשם SIP (קיצור שם System Integrity Protection, ידוע גם כ-rootless) מונע הדבקה של מערכת ההפעלה עצמה, כולל:

- מניעת כתיבה של קבצים לאזורים מוגנים במערכת ההפעלה (בדרך כלל תחת System/ אבל לא רק).
- מונע טעינת דרייברים (kernel extensions ב-macOS) שלא אושרו מראש על ידי אפל.
- מונע דיבוג של תהליכים שחתומים על ידי אפל.
- מונע kernel debugging, כולל DTrace.
- מונע שינוי של משתני NVRAM.

כפי שניתן להתרשם, SIP הוא מנגנון הגנה חזק מאד. מערכות macOS מגיעות עם SIP דלוק כברירת מחדל, והדרך הלגיטימית היחידה לכבות אותו היא לעשות reboot למצב מיוחד הנקרא Recovery OS שממנו ניתן לכבות את SIP - לא פעולה שניתנת לאוטומציה וגם לא משהו שמשתמש קצה "רגיל" צפוי לעשות.

מנגנון SIP עצמו מתוחזק על ידי הקרנל ומושפע ממשתני NVRAM (לכן ההגנה על משתני NVRAM הכרחית, אגב) וספציפית ממשתנה בשם csr-active-config. שווה לציין שעקיפה של כל מנגנון אכיפה שהוזכר עלול

להפיל את כל SIP (כתיבת משתני NVRAM באופן טריוויאלי, אבל גם דברים אחרים כגון טעינת דרייברים, דיבוג של תהליכים חתומים או אפילו כתיבה על קבצים מוגנים).

מבין כל האכיפות הללו, האכיפה המורגשת ביותר על ידי משתמשי קצה הוא מניעת כתיבה של קבצים לאזורים מוגנים, והיסטורית - חלק גדול מעקיפות SIP התמקדו באזור זה.

נקודה חשובה להמשך המאמר היא שחברת אפל נתקלת בבעייה - בעת עדכון, מערכת ההפעלה צריכה לדרוס קבצים שנמצאים תחת אזורים מוגנים ולכן צריכה בעצמה לעקוף את SIP! לשם כך, הוגדרו תהליכים שחתומים על ידי אפל ועבורם מוגדרות יכולות השמורות רק להם - אותן יכולות ידועות כ-Entitlements. ישנם Entitlements רבים (למעשה, ישנו Database בו הם מתוחזקים) אך אנחנו מעוניינים בשניים מאד ספציפיים:

1. com.apple.rootless.install - בינארי המחזיק ב-entitlement זה יכול לעקוף את אכיפת SIP על מערכת הקבצים.
2. com.apple.rootless.install.heritable - בינארי המחזק ב-entitlement זה מוריש לבניו את com.apple.rootless.install.

באופן טבעי, בינארים אלו הם ה-"קורבנות הטבעיים" של חולשת מעקף SIP, והיום נתמקד באחד מהם.

## גילוי החולשה

צוות המחקר של Microsoft Defender ביצע חיפוש שגרתי של קוד זדוני על מערכות macOS וגילה הרצה של בינארי בשם drop_sip. אותו בינארי נראה נפוץ מאד (המון הרצות על תחנות קצה רבות) - ושמו מרמז על מעקף של SIP. הצוות בחן את אותו בינארי ונראה שהוא **מאפשר מחדש אכיפת SIP על מערכת הקבצים** ולאחר מכן פשוט מריץ שורת פקודה שהוא מקבל בארגומנטים):

```
__int64 __fastcall start(__int64 argc, char **argv, char *const *envp)
{
  const char *err_msg_fmt; // r14
  const char *prog_name; // rax

  if ( csops(0LL, 12LL, 0LL, 0LL) )
  {
    err_msg_fmt = "%s: Failed to clear flag.\n";
  }
  else
  {
    execve(argv[1], argv + 1, envp);
    err_msg_fmt = "%s: Failed to exec.\n";
  }
  prog_name = getprogname();
  fprintf(*&_mh_execute_header.magic, err_msg_fmt, prog_name);
  return 1LL;
}
```

הקריאה csops היא syscall פרטי של אפל, והמספר 12 שמור כקבוע CS_OPS_CLEARINSTALLER - אותו קבוע "מנקה" את com.apple.rootless.install מהתהליך.

הגילוי כי drop_sip הוא בינארי סטנדרטי ולא מזיק היה גילוי טוב (הצוות חושב שאפל צריכים לשנות את השם ל-"drop_drop_sip") אך מעלה סוגייה מעניינת - מדוע ש-drop_sip יניח מראש שהוא יכול לעקוף אכיפת SIP על מערכת הקבצים?

התשובה היא ש-drop_sip הוא תהליך בן של תהליך בשם systemmigrationd, ואותו תהליך אכן מחזיק ב-com.apple.rootless.heritable:

```
root@McJbo Desktop # codesign -dvv --entitlements - /System/Library/PrivateFrameworks/systemmigrationd | grep rootless
Executable=/System/Library/PrivateFrameworks/SystemMigration.framework/Versions/A/Resources/systemmigrationd
Identifier=com.apple.systemmigrationd
Format=Mach-O universal (x86_64 arm64e)
CodeDirectory v=20400 size=755 flags=0x0(none) hashes=13+7 location=embedded
Platform identifier=14
Signature size=4442
Authority=Software Signing
Authority=Apple Code Signing Certification Authority
Authority=Apple Root CA
Signed Time=Dec 4, 2022 at 8:00:06 PM
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=76
        [Key] com.apple.rootless.volume.Preboot
        [Key] com.apple.rootless.install.heritable
        [Key] com.apple.rootless.datavault.controller
root@McJbo Desktop #
```

התהליך systemmigrationd הוא daemon שאחראי ל-Migration ("ניוד" של מידע בין macOS ל-macOS ואפילו מ-Windows). מכיוון ש-systemmigrationd מחזיק ב-entitlement חזק כל כך תהינו אילו עוד תהליכים רצים תחתיו, ומהר מאד גילינו שניים מעניינים: bash ו-perl. הסיבה לכך ששניהם מעניינים הם כי הם interpreters, ולכן קל יחסית לגרום להם להריץ קוד כרצוננו (הזרקות אינו עניין פשוט ב-macOS, גם כאשר רצים כ-root, והסיבה לכך נעוצה גם באכיפות שונות על ידי SIP, בין היתר). ובכן, מהר מאד גילינו שניתן להשפיע על perl ועל bash בקלות, עם משתני סביבה:

- perl יחפש משתנה סביבה בשם PERL5OPT ומכיל command-line-switches שיכולים להריץ פקודות perl נוספות.
- bash יחפש משתנה סביבה בשם BASH_ENV שיכול להחזיק פקודות bash להריץ בעת עלייה במצב לא אינטרקאטיבי.

ב-macOS הדרך לקבוע משתני סביבה תחת launchd (תהליך האב של הכל ב-macOS, כולל של daemons כגון systemmigrationd) היא עם launchctl. הנה דוגמא לקביעת משתנה סביבה שיריץ את הקובץ /private/tmp/migraine.sh בכל פעם ש-perl מתחיל לרוץ:

```
launchctl setenv PERL5OPT '-Mwarnings;system("/private/tmp/migraine.sh")
```

ובכן, הרצת Migration מתאימה אכן גורמת לקוד שלנו לרוץ! בדוגמא שלנו דרסנו את הקובץ /Library/Apple/System/Library/Extensions/AppleKextExclusionList.kext/Contents/Resources/ExceptionLists.plist - זהו קובץ שמוגן על ידי SIP ומכיל מידע על Kernel extensions שניתנים לטעינה.

דריסה שכזו יכולה לאפשר טעינת Kernel extensions משלנו, אבל אנחנו רק כתבנו בקובץ Migraine לצורך ההדגמה:

```
mipearse@Mikes-MBP ~ % csrutil status
System Integrity Protection status: enabled.
mipearse@Mikes-MBP ~ % cat /Library/Apple/System/Library/Extensions/AppleKextExcludeList
.kext/Contents/Resources/ExceptionLists.plist
Migraine
mipearse@Mikes-MBP ~ % ls -laO /Library/Apple/System/Library/Extensions/AppleKextExclude
List.kext/Contents/Resources/ExceptionLists.plist
-rw-r--r--  1 root  wheel  restricted 9 Mar  3 13:52 /Library/Apple/System/Library/Exten
sions/AppleKextExcludeList.kext/Contents/Resources/ExceptionLists.plist
mipearse@Mikes-MBP ~ %
```

## אוטומציה

בשלב זה כבר יכולנו לדווח לאפל על החולשה, אבל רצינו לבצע אוטומציה, שכן הרצת Migration (המתבצעת באמצעות אפליקציה בשם Migration Assistant ) היא תהליך ידני מאד שגם גורם לניתוק המשתמשים המחוברים - כלומר, החולשה שמצאנו טובה מספיק רק עבור תוקף עם גישה פיזית. לכן, החלטנו להשקיע מחקר נוסף ולבדוק האם ניתן לגרום לחולשה לרוץ ללא גישה פיזית, בהנחה שהתוקף רץ מרחוק. להלן רצף האירועים בביצוע Migration:

א. אפליקציית ה-Migration Assistant משתמשת בכלי בשם Setup Assistant לצורך התחלת הניוד, כאשר תהליך בשם MBSystemAdministration מתווך ביניהם על ידי שימוש ב-XPC. כמו כן, ניתוק המשתמשים הפעילים מתבצע בשלב זה על ידי קריאה למתודה בשם :

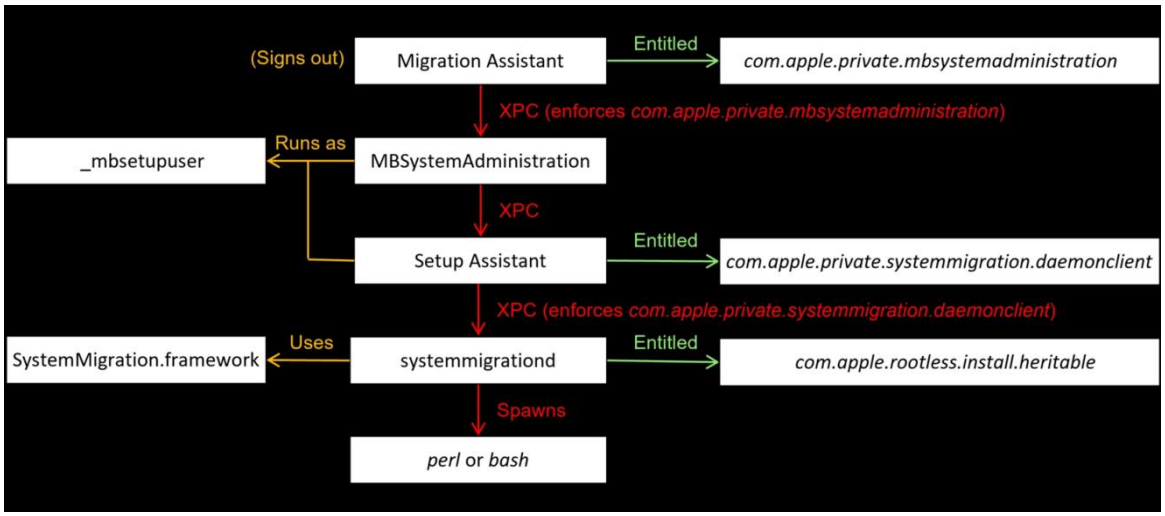
SACLOStartLogoutWithOptions

ב. כפי שציינו, תהליך ה-MBSystemAdministration מתווך בין Migration Assistant ו-Setup Assistant. תהליך זה מוודא שהתהליך הקורא (Migration Assistant במקרה שלנו) מחזיק ב-Entitlement בשם com.apple.private.mbsystemadministration - אחרת הוא יסרב לשרת את הבקשה. באופן מעניין, מכיוון שכל המשתמשים הפעילים נותקו בשלב זה, MBSystemAdministration רץ בתור משתמש חבוי בשם _mbsetupuser המאפשר ביצוע פעולות UI גם לאחר שכל המשתמשים נותקו.

ג. כלי ה-Setup Assistant מקבל בקשות דרך MBSystemAdministration ומבצע בקשות XPC בעצמו לשירותי Mach שונים שממומשים על ידי systemmigrationd, שזהו בדיוק ה-daemon שיכול לעקוף אכיפת SIP. אותו systemmigrationd מוודא שהתהליך הקורא (Setup Assistant) מחזיק ב-Entitlement בשם com.apple.private.systemmigration.daemonclient - אחרת הוא יסרב לבקשה.

כדאי לציין ש-systemmigrationd גם הוא, עושה שימוש רבות ב-Private Framework בשם SystemMigration.framework, ואותו framework מממש שרת הממתין לבקשות (המימוש הוא במתודה בשם startListeningForConnections). הבקשות מגיעות על גבי מערכת הקבצים תחת התיקיה /Library/SystemMigration/Queue (המוגנת על ידי SIP). קבצים בתיקיה זו מתארים את בקשת הניוד, ולאחר ניטור ישנו את שמם לקובץ בשם "In-Flight". זהו השלב בו הניוד עצמו מתבצע, כולל קריאה לסקריפטים שירוצו על ידי perl או bash.

להלן תיאור סכמטי המסכם את תהליך הניוד:




אם כן, המטרה היא לדלג על ביצוע ההתנתקות על ידי Migration Assistant. הנסיונות הראשוניים שלנו כללו בעיקר ביצוע Patching מסוגים שונים, אך ללא הצלחה - למשל, ביצוע Patching ל-SACLOStartLogoutWithOptions למשל יגרום לקריסה כמעט מיידית עקב Pointer Authentication Code. נסיונות דומים גם לא צלחו - למשל, ב-macOS בינארים רבים מכילים כמה גרסאות המקומפלים לארכיטקטורות שונות (למשל Intel ו-ARM), ושליפת הגרסא המתאימה לארכיטקטורה שעליה רצים גורמת באופן דומה לכשלון (הפעם הסיבה היא ש-Migration Assistant מאבד את ה-Entitlement הדרוש עבורו).

רעיון נוסף שעלה לצוות הוא בחינת שרשרת האירועים באופן מדוקדק יותר - גילינו כי Setup Assistant רץ עם ארגומנט MiniBuddyYes-, ותהינו מהי משמעותו. התקווה שלנו היא שנוכל להריץ את Setup Assistant ישירות ועל ידי כך לעקוף לגמרי את ניתוק המשתמשים הפעילים.

בחינת המימוש של Setup Assistant חשפה מספר דגלים מעיינים:

```
54      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-MiniBuddyYes")) )
55      {
56        v15 = self;
57        v16 = "setPrimaryBuddyType:";
58        goto LABEL_12;
59      }
60      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-MigrationBuddyYes")) )
61      {
62        v17 = self;
63        v18 = "setPrimaryBuddyType:";
64 LABEL_23:
65        objc_msgSend(v17, v18, 2LL);
66        goto LABEL_13;
67      }
68      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-EOSBuddyYes")) )
69      {
70        -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 3LL);
71        goto LABEL_13;
72      }
73      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-ResumeBuddyYes")) )
74      {
75        v17 = self;
76        v18 = "setSecondaryBuddyType:";
77        goto LABEL_23;
78      }
79      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-XCUITest")) )
80      {
81        firstLoginContext = self->firstLoginContext;
82        self->firstLoginContext = (NSString *)CFSTR("kShouldUseTestValues");
83        objc_release(firstLoginContext);
84      }
```

בנוסף, גילינו מתודה בשם useDebugParameters שאף היא מחפשת דגל (הפעם בשם MBDebug-):

```
1  bool __cdecl -[MacBuddyAppDelegate useDebugParameters](MacBuddyAppDelegate *self, SEL a2)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    if ( getuid() )
6      return 0;
7    v4 = +[NSProcessInfo processInfo](&OBJC_CLASS___NSProcessInfo, "processInfo");
8    v5 = objc_retainAutoreleasedReturnValue(v4);
9    v6 = -[NSProcessInfo arguments](v5, "arguments");
10   v7 = objc_retainAutoreleasedReturnValue(v6);
11   objc_release(v5);
12   v8 = (unsigned int)-[NSArray containsObject:](v7, "containsObject:", CFSTR("-MBDebug"));
13   v3 = v8;
14   if ( !v8 )
15     goto LABEL_30;
16   v29 = v8;
17   -[MacBuddyAppDelegate setRunningAsDebug:](self, "setRunningAsDebug:", 1LL);
18   -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 0LL);
19   -[MacBuddyAppDelegate setSecondaryBuddyType:](self, "setSecondaryBuddyType:", 0LL);
20   v33 = 0u;
21   v34 = 0u;
22   v31 = 0u;
23   v32 = 0u;
24   v30 = v7;
25   v9 = objc_retain(v7);
26   v10 = -[NSArray countByEnumeratingWithState:objects:count:](
27           v9,
28           "countByEnumeratingWithState:objects:count:",
29           &v31,
30           v35,
31           16LL);
32   if ( !v10 )
33     goto LABEL_27;
34   v11 = v10;
     v12 = *(_OWORD *)v32;
```

באופן מפתיע, הרצה של Setup Assistant עם הדגל MBDebug- אכן גורמת לניוד ללא ניתוק! דגל נוסף בשם ResumeBuddyYes- מדלג אפילו על כמה שלבי UI. מכיוון שעדיין תוקף צריך לבצע שלבי UI, השתמשנו ביכולות אוטומציה (עם Apple Script) כדי לבצע לחיצות אוטומאטיות (בדומה ל-AutoIT ב-Windows). לפיכך, ההשמשה פשוטה:

1. הכנה של Time Machine Backup בגודל של 1GB, לצורך ניוד, וביצוע mount שלו עם hdiutil.
2. הכנה של payload שירוץ ללא הגבלות SIP. בהדגמה שלנו כתבנו פשוט את המילה Migraine לתוך קובץ, אבל ניתן להריץ כל דבר.
3. הגדרת משתנה הסביבה PERL5OPT באמצעות launchctl על מנת לגרום ל-payload שלנו לרוץ כאשר perl מתחיל.
4. הרצת Setup Assistant עם הדגלים MBDebug- ו-ResumeBuddyYes-.
5. הרצת Apple Script שמבצע באופן אוטומאטי לחיצות, כך שנבחרת האפשרות " From a Mac, Time Machine backup or Startup Disk" ולאחר מכן לחיצה על Continue מספר פעמים.

## השלכות החולשה ותיקונה

לעקיפת SIP יש השלכות משמעותיות על מערכת ההפעלה, שכן תוקף יכול לבצע פעולות מסוכנות רבות:

א. יצירת קבצים בלתי ניתנים למחיקה: זוהי ההשלכה הישירה ביותר ומעניינת מאד יצרני Endpoint Protection כגון Microsoft Defender. קבצים המוגנים על ידי SIP בלתי ניתנים למחיקה באופן אוטומאטי, ועם המעבר של אפל מ-Kernel Extensions לתשתית בשם Endpoint Security Framework, באופן פרקטי לא קיים פתרון לבעיה שבה malware עוקף SIP.

ב. הרחבת משטח התקיפה באופן משמעותי, כולל ל-kernel: המאמר המעולה של Mickey Jin מתאר כיצד ניתן להריץ קוד kernel בהנתן עקיפת SIP. באופן דומה לנקודה הקודמת, ההשלכה של מעבר ל-Endpoint Security Framework היא שקוד זדוני שרץ ב-kernel בלתי ניתן לניטור הולם.

ג. יצירת rootkit: תוקף שיכול לטעון קוד kernel יכול לממש rootkit על כך המשתמע מכך - החבאת קבצים ותהליכים, החבאת חיבורי רשת ועוד. ספציפית, Microsoft Defender מכילה פיצ'ר בשם Tamper Protection המגן על התהליכים והקבצים הרלוונטיים עבורו, והחשש הוא עקיפת ה- Tamper Protection, למשל.

ד. עקיפת TCC: בנוסף לטעינת קוד kernel, המאמר של Mickey Jin מתאר כיצד ניתן לעקוף מנגנון הגנה אחר בשם TCC. מנגנון זה אחראי להגנה על פרטיות משתמש הקצה, כולל גישה לקבצים פרטיים, למצלמה, למיקרופון, לשירותי מיקום ועוד.

## סיכום

פנינו לחברת אפל באופן דיסקרטי ולאחר מספר חודשים החולשה תוקנה תחת CVE-2023-32369. ב-18 במאי 2023. אנחנו מודים לאפל על התיקון וממליצים לכל המשתמשים לוודא שמערכת ההפעלה שלהם מעודכנת.

באפריל 2023, לאחר התיקון ב-Beta, הצוות שיתף פעולה עם אפל לוודא שהחולשה נסגרה באופן הרמטי. מעניין לגלות מספר תיקונים משמעותיים מאד:

1. אפל החליטה לחסום לגמרי שינוי global environment variables כאשר SIP דלוק. באופן כללי לא ניתן להשפיע על המשתנים של launchd. זהו פתרון "ברוטאלי" אבל הוא פותר באופן אפקטיבי מאד מחלקה שלמה של חולשות המתבססות על הרעלה של משתני סביבה.

2. אפל הוסיפה Launch Constraint על systemmigrationd כך שלא ניתן לטעון אותו באמצעות Launch Daemon שונה מההגדרה המקורית. הסיבה לכך היא שקבצי plist המגדירים launch daemons מכילים גם משתני סביבה הניתנים להגדרה מלאה, והתוכנית של הצוות הייתה להגדיר משתנה סביבה עבור systemmigrationd. זהו איננו מאמר על Launch Constraints - לרקע מומלץ לקרוא סיכום ראשוני על ידי Csaba Fitzl. באופן דומה, גם פתרון זה פותר מחלקה גדולה של חולשות פוטנציאליות.

3. המימוש של systemmigrationd כעת מסיר את יכולות עקיפת ה-SIP לפני הרצת סקריפטים (בדומה לדרך שבה drop_sip עובד). זהו תיקון טקטי אך משמעותי - הצוות הצליח עדיין לעקוף את הפתרונות האחרים על ידי שינוי dependencies של perl ללא שינוי משתני סביבה, אך כאמור - עקב תיקון טקטי זה הצוות לא הצליח לבצע עקיפת SIP.

צוות המחקר מאמין שהחולשה נסגרה באופן הרמטי על ידי אפל, יחד עם צמצום משמעותי של משטח התקיפה על SIP.

## על הכותב

יונתן בר אור (Jonathan Bar Or, "JBO") חוקר בחברת מייקרוסופט תחת Microsoft Defender ובאופן רשמי ארכיטקט המחקר למערכות הפעלה שאינן Windows.

כותב פה ושם בטוויטר תחת @yo_yo_yo_jbo.