Technologic Papers

אימות מצביעים (Pointer Authentication)

מאת עידן רוזנצוייג

הקדמה

חולשות שיבוש זיכרון (Memory Corruption) הן סוג החולשות הנפוץ והמסוכן ביותר. הן מאפשרות לתוקפים לזייף או להחליף מצביעים ולהשתלט על נתיב זרימת התוכנית (Control Flow) באמצעות טכניקות כגון Return Oriented Programming (ידוע בתור ROP) או Jump Oriented Programming (ידוע בתור JOP). הגנות מודרניות - הכוללות ASLR, Stack Canaries, DEP מספקות הגנה טובה, אך לעיתים קרובות אינן מספיקות מול תוקפים מתוחכמים המסוגלים להדליף כתובות, לשרשר גאדג'טים (gadgets) או לנצל טקטיקות דומות.

אימות מצביעים מציע מענה מבוסס קריפטוגרפיה שתוכנן ספציפית כדי לצמצם זיוף ושיבוש של מצביעים. מאמר זה בוחן את היסודות התיאורטיים של אימות מצביעים, את המימוש הקונקרטי שלו בארכיטקטורת ARM, דוגמאות מעשיות, פגיעויות ומתקפות מהעולם האמיתי, ועוד.

המאמר לא דורש ידע מקדים רחב, אך מומלץ להכיר את הקונספט של אימות קריפטוגרפי, השימושים בו, והתקפות שונות שיכולות להיות כנגד אימות קריפטוגרפי. מומלץ גם להכיר את ארכיטקטורת ARM64.

תיאוריית אימות מצביעים

הרעיון המרכזי מאחורי אימות מצביעים הוא להוסיף אימות קריפטוגרפי למצביעים כדי למנוע זיוף ושיבוש. אימות מצביעים עובד על ידי שיוך תג אימות (חתימה) למצביע, ובדיקת התג בעת שימוש במצביע.

תהליך חישוב התג מכונה לרוב **חתימה (Signing)**, ואילו התהליך ההפוך של אימות התג נקרא **אימות (Authenticating).**

המנגנון הקריפטוגרפי ושאר המפתחות השייכים לו מכונים יחד **סכמת חתימה (Signing scheme)**. סכמת החתימה חייבת לעמוד במטרות האבטחה הבאות:

- מניעת זיוף: סכמת החתימה חייבת למנוע מתוקפים את האפשרות לחתום מצביעים לבחירתם.
- הגנה נגד known plaintext attacks: גישה למאגר גדול של מצביעים חתומים לא יעזור לתוקפים להדליף שום מפתח פרטי פנימי של המנגנון או לחזות תגים חוקיים למצביעים.
- הגנה נגד מתקפות החלפה (substitution attacks): סכמת החתימה חייבת למנוע מתוקפים להחליף בין מצביעים חוקיים.

על מנת להשיג מטרות אבטחה אלו, סכמת החתימה משתמשת במנגנון קריפטוגרפי חזק, ומחשבת כל תג בעזרת שילוב רחב של קלטים אפשריים שנגישים גם לצד שחותם וגם לצד שמאמת. חלק מקלטים אלו נבחרים בגלל היותם חסינים כנגד הדלפות או חיזוי, ובכך מחזקים את ההגנה נגד זיוף, וקלטים אחרים נבחרים בעיקר בשביל גיוון בכדי למנוע מתקפות החלפה (substitution attacks):

- מפתח גלובלי רנדומלי סודי
- המיקום של המצביע עצמו בזיכרון
- המיקום שהמצביע מפנה אליו (הערך של המצביע)
- ערך המחסנית
- סוג המצביע
- האינדקס של המצביע בטבלת הוירטואלית (vtable).
- ה-decleration/name של הפונקצייה/משתנה של המצביע
- כל ערך קבוע

(שימו לב שלא כל הקלטים רלוונטיים בכל המקרים, אלה רק דוגמאות)

אידיאלית, כל use case שונה למצביע אמור לקבל סכמת חתימה ייחודית יחד איתו, בכדי להפריד לגמרי use case שונים, ולמנוע מתקפות החלפה (substitution attacks).

עדיף להשתמש בקלטים אשר לתוקפים יהיה קשה להדליף או לחזות, תלוי ב-use case הספציפי. ככל שסכמת חתימה מגוונת יותר (מבוססת על יותר קלטים), ככה היא טובה יותר נגד זיופים, מכיוון שיש יותר קלטים שתוקפים צריכים להדליף או לחזות.

מימושים של אימות מצביעים

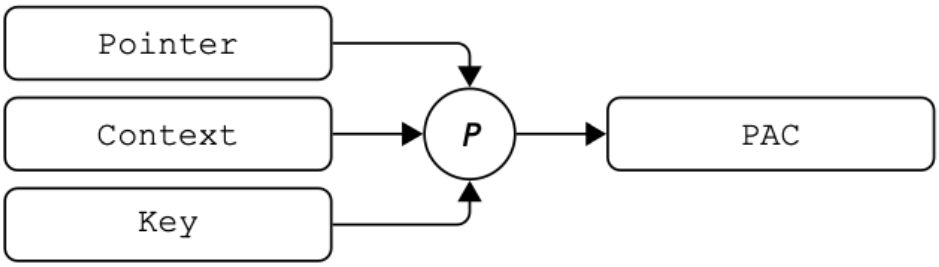
ארכיטקטורת ARM

ARM הוסיפו אימות מצביעים לראשונה בשנת 2016 בארכיטקטורת ה-ARMv8.3-A. הם קראו לזה **Pointer Authentication Code** או **PAC**. ארכיטקטורה זאת וכל הבאות אחריה גם מממשות PAC.

בארכיטקטורת ARM, כל תג מחושב בעזרת אלגוריתם תלוי-מימוש שמקבל:

- ערך המצביע
- כל אחד מהמפתחות הגלובאליים: ישנם 5 מפתחות גלובליים סודיים שמוגדרים ונשלטים על ידי רמות הרשאה גבוהות יותר. המפתחות הם:
  - מפתח לקוד A (נקרא IA key)
  - מפתח לקוד B (נקרא IB key)
  - מפתח למידע A (נקרא DA key)
  - מפתח למידע B (נקרא DB key)
  - מפתח כללי
- ערך הקשר: יכול להיות כל ערך שרוצים. הערך הזה לרוב מורכב משילוב של כמה קלטים, כמו ברשימה בתיאוריית אימות מצביעים (מלבד המפתח הגלובאלי, שהוא כבר מובנה בחלק החישוב)

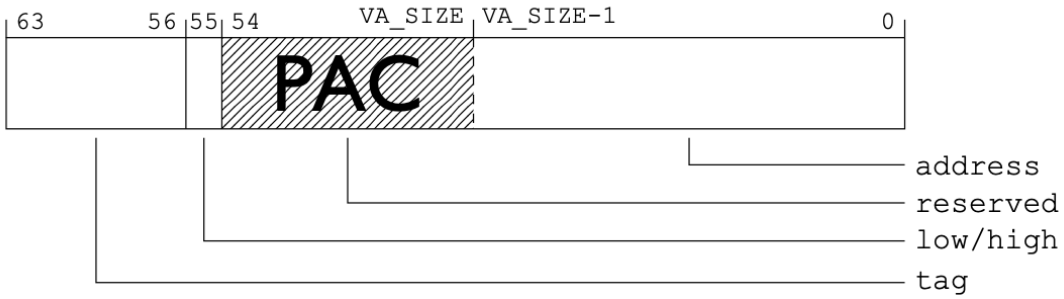
הנה דיאגרמה פשוטה שמראה איך התג מחושב:



ערך התג נשמר בחלק הלא משומש של מילת המצביע (הביטים במקומות הגבוהים). הסיבה שיש חלק לא משומש בכל מילת מצביע היא בגלל שהערך של מצביעים מוגבל לגודל הזיכרון האמיתי שקיים במערכת, שבדרך כלל הרבה יותר קטן מהגודל המקסימלי שמילת מצביע תומכת (מה שמשאיר חלק לא משומש בכל מילת מצביע). זה גם אומר שהתג יכול להיות בגדלים שונים במערכות שונות.

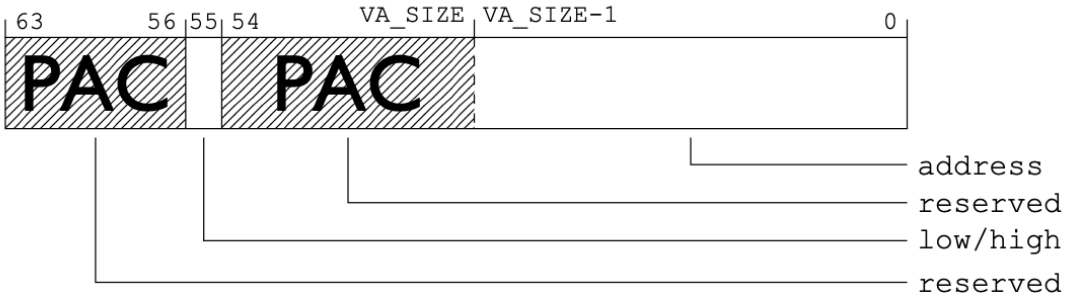
זה המבנה של מילת מצביע במערכת שבה ערך המצביע מוגבל ל-48 ביטים, עם tagging מופעל (עוד פיצ'ר ב-ARM). לתג יש גודל של 7 ביטים:

... e.g. 7 bits with 48-bit VA with tagging
... leaving remaining bits intact



זה המבנה של מילת מצביע במערכת שבה ערך המצביע מוגבל ל-48 ביטים, בלי tagging (עוד פיצ'ר בARM). לתג יש גודל של 15 ביטים:

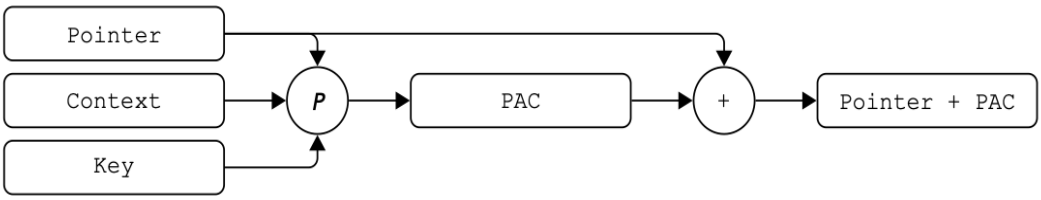
... e.g. 15 bits with 48-bit VA without tagging
... leaving remaining bits intact



חלק מהפקודות ב-ARM שקשורות ל-PAC:

פקודות שחותמות מצביעים, הן מחשבות את התג באמצעות הארגומנטים הניתנים להן, ושומרות את התג בחלק הלא משומש של מילת המצביע:

```
PAC*
```



חתום את המצביע באוגר Xd עם המפתח IA או IB, עם ערך ההקשר באוגר Xn:

```
PACIA Xd, Xn
PACIB Xd, Xn
```

חתום את המצביע באוגר Xd עם המפתח IA או IB, עם ערך הקשר 0:

```
PACIZA Xd
PACIZB Xd
```

חתום את המצביע באוגר x17 עם המפתח IA או IB, עם ערך ההקשר באוגר x16:

```
PACIA1716
PACIB1716
```

חתום את המצביע באוגר x30 (אוגר הקישור - link register) עם המפתח IA או IB, עם ערך ההקשר באוגר sp (אוגר המחסנית):

```
PACIASP
PACIBSP
```

חתום את המצביע באוגר x30 (אוגר הקישור - link register) עם המפתח IA או IB, עם ערך הקשר 0:

```
PACIAZ
PACIBZ
```

פקודות שמאמתות מצביעים, הן בודקות את התג השמור בחלק הלא משומש של מילת המצביע. אם האימות מצליח, התג מוסר מהחלק הלא משומש של מילת המצביע; אם הוא נכשל, הפקודה משבשת את המצביע (משבשת את החלק הלא משומש של מילת המצביע) כך שכל שימוש עתידי בו יהיה לא חוקי:

```
AUT*
```

אמת את המצביע באוגר Xd עם המפתח IA או IB, עם ערך ההקשר באוגר Xn:

```
AUTIA Xd, Xn
AUTIB Xd, Xn
```

אמת את המצביע באוגר Xd עם המפתח IA או IB, עם ערך הקשר 0:

```
AUTIZA Xd
AUTIZB Xd
```

אמת את המצביע באוגר x17 עם המפתח IA או IB, עם ערך ההקשר באוגר x16:

```
UTIA1716
AUTIB1716
```

אמת את המצביע באוגר x30 (אוגר הקישור - link register) עם המפתח IA או IB, עם ערך ההקשר באוגר sp (אוגר המחסנית):

```
AUTIASP
AUTIBSP
```

אמת את המצביע באוגר x30 (אוגר הקישור - link register) עם המפתח IA או IB, עם ערך הקשר 0:

```
AUTIAZ
AUTIBZ
```

פקודות שמסירות את התג מהחלק הלא משומש של מילת המצביע, בלי לבצע אימות:

```
XPAC*
```

הסר את התג ממצביע של קוד באוגר Xd:

```
XPACI Xd
```

הסר את התג ממצביע של מידע באוגר Xd:

```
XPACD Xd
```

הסר את התג מהמצביע באוגר x30 (אוגר הקישור - link register):

```
XPACLRI
```

פקודות משולבות המשמשות הן לאימות והן לביצוע קפיצה:

```
*BLRA* BRA
```

אמת את המצביע באוגר Xn עם המפתח IA או IB, עם ערך ההקשר באוגר Xm. לאחר מכן קפוץ לכתובת זו:

```
BRAA Xn, Xm
BRAB Xn, Xm
```

אמת את המצביע באוגר Xn עם המפתח IA או IB, עם ערך ההקשר באוגר Xm. לאחר מכן בצע קפיצה עם קישור (branch with link) לכתובת זו:

```
BLRAA Xn, Xm
BLRAB Xn, Xm
```

אמת את המצביע באוגר Xn עם המפתח IA או IB, עם ערך הקשר 0. לאחר מכן קפוץ לכתובת זו:

```
BRAAZ Xn
BRABZ Xn
```

אמת את המצביע באוגר Xn עם המפתח IA או IB, עם ערך הקשר 0. לאחר מכן בצע קפיצה עם קישור (branch with link) לכתובת זו:

```
BLRAAZ Xn
BLRABZ Xn
```

פקודות משולבות המשמשות הן לאימות והן לביצוע חזרה:

```
RETA*
```

אמת את המצביע באוגר x30 (אוגר הקישור - link register) עם המפתח IA או IB, עם ערך ההקשר באוגר sp (אוגר המחסנית). לאחר מכן בצע חזרה רגילה:

```
RETAA
RETAB
```

חברת Apple הכניסה את ארכיטקטורת ARMv8.3-A לסדרת מעבדי ה-Apple Silicon שלה באזור שנת 2018:

- A-Series: כל מעבד החל מ-A12 והלאה תומך בארכיטקטורת ARMv8.3-A (לפחות).
- M-Series: כל מעבד בסדרה זאת תומך בארכיטקטורת ARMv8.3-A (לפחות).
- S-Series: כל מעבד החל מ-S4 והלאה תומך בארכיטקטורת ARMv8.3-A (לפחות).

מימוש ה-PAC של Apple משתמש בגרסה מיוחדת של צופן הבלוק QARMA בתור אלגוריתם הבסיס.

מימוש מבוסס תוכנה

למרות שזה לא נפוץ במיוחד, ניתן בהחלט לממש אימות מצביעים גם בתוכנה. במקום פקודות ייעודיות ברמת החומרה, תהליכי החתימה והאימות ממומשים כקטעי קוד קטנים (stubs) או כפונקציות inline.

דוגמאות קוד לאימות מצביעים

ארכיטקטורת ARM

הגנה על כתובת החזרה (Return Address) במחסנית הקריאות:

נרצה להגן על המצביע של כתובת החזרה שבמחסנית הקריאות (Call Stack). נחתום את כתובת החזרה עם הכניסה לפונקציה, ונאמת אותה לפני היציאה ממנה.

קיימים מספר קלטים אפשריים שנגישים הן בכניסה לפונקציה והן ביציאה ממנה:

- מצביע המחסנית (stack pointer): ערך זה זהה בכניסה לפונקציה וביציאה ממנה.
- מצביע המסגרת (frame pointer): ערך זה קבוע לאורך ריצת הפונקציה.
- מצביע המסגרת הקודם: ערך זה קבוע לאורך ריצת הפונקציה.

במימוש טיפוסי של מחסנית קריאות ב-ARM, כתובת החזרה נחתמת על בסיס המפתח הסודי IB ומצביע המחסנית:

```
; prologue
PACIBSP ; sign the pointer in register x30 (the link register) with the key IB,
with the context value in register sp (the stack pointer)
STP X29, X30, [SP, #-16]! ; push the previous function frame pointer and the
current link register to the stack
MOV X29, SP ; setup the frame pointer for the current function

; ... function body ...

; epilogue
```

```
LDP X29, X30, [SP], #16 ; restore the previous function frame pointer and the
current link register from the stack
AUTIBSP ; authenticate the pointer in register x30 (the link register) with the
key IB, with the context value in register sp (the stack pointer).
RET ; return to the pointer in register x30 (the link register)
```

הגנה על מצביעי Callback:

נרצה להגן על מצביעי Callback. נחתום על המצביע בעת הרישום (אחסון) שלו, ונאמת אותו לפני שנקרא לו.

אין הרבה קלטים אפשריים שנגישים הן לצד החותם והן לצד המאמת. אם המצביע מאוחסן במשתנה גלובלי (או בהקשר גלובלי דומה), ניתן להשתמש בכתובת שלו כקלט.

במימוש פשוט של מצביע Callback ב-ARM, המצביע נחתם על בסיס המפתח הסודי IA וערך קבוע כלשהו:

```
LDR X17, [X19, #0x88]   ; load the signed callback ptr
; authenticate the signed callback with the key IA, combined with a constant
value, and then branch with link
MOV X16, #0xAF32
BLRAA X17, X16
```

הגנה על setjmp/longjmp:

נרצה להגן על מצביע היעד (the goto address) במנגנון setjmp/longjmp. נחתום את מצביע היעד בעת יצירת אובייקט ה-jmp (ב-setjmp), ונאמת אותה בעת הקפיצה אליו (ב-longjmp).

ישנם קלטים רבים שנגישים הן בעת היצירה והן בעת השימוש (באופן כללי, כל המטא-דאטה באובייקט ה-jmp):

- מצביע המחסנית
- ערכי אוגרים
- מידע נוסף של ריצת התהליך/תהליכון בזמן יצירת אובייקט ה-jmp

במימוש טיפוסי של setjmp/longjmp ב-ARM, כתובת היעד נחתמת על בסיס המפתח הסודי IA ומצביע המחסנית:

```
; setjmp
MOV X0, SP
PACIA X30, X0 ; sign the return address (for the setjmp) with the key IA, with
the context value in register x0
STR X30, [X1, #0] ; save the signed return address to the setjmp obj
STR X0, [X1, #8] ; save the sp itself to the setjmp obj
; ... rest of setjmp store ...
```

```
; longjmp
LDR X30, [X1, #0] ; load the saved return address
LDR  X0, [X1, #8] ; load the saved stack pointer
AUTIA X30, X0 ; authenticate the return address with the key IA, with the
context value in register x0
MOV SP, X0 ; restore the stack pointer
; ... rest of longjmp restore ...
RET X30 ; branch back to the return address
```

אימות מצביעים בחיים האמיתיים

דמיינו פארק רכבות הרים שבו, בנוסף לכרטיס הרגיל, ניתן לרכוש גם כרטיס VIP. כרטיס ה-VIP פועל באופן הבא:

- אתם ניגשים לכניסה למתקן. הסדרן רושם תווית על פתק המציינת שאתם זכאים להיכנס למתקן הזה בזמן יעד מסוים בעתיד (או מאוחר יותר) ולהיכנס מיד, מבלי להמתין בתור. התווית מכילה:
  - שם המתקן
  - זמן היעד
  - חתימת מתקן ייחודית, חלקה בלתי נראה ובלתי ניתן לזיוף
  - מספר המתקנים היומי שלכם (שעולה בערכו עבור התווית הבאה)
  - סיכום ביקורת (Checksum) של התווית (על כל הפרטים המופיעים בה)
- כאשר אתם מגיעים למתקן, הקופאי מחשב מחדש את סיכום הביקורת על התווית. אם התווית מאומתת בהצלחה וזמן היעד תקף, הקופאי מממש את התווית (משמיד אותה) ומאפשר לכם להיכנס למתקן.
- אתם יכולים, כמובן, להגיע למתקן מוקדם יותר ולבטל את התווית.
- מותר לכם להשתמש בתווית אך ורק עבור אותו מתקן שאליו נרשמתם.
- לא ניתן לרשום מספר תוויות במקביל - רק תווית אחת בכל פעם.
- אין הגבלה על המספר הכולל של תוויות שתרשמו לאורך היום.

ניתן לחשוב על המודל הזה כאל מצביעים חתומים למתקנים. אינכם יכולים פשוט לזייף מצביע, כיוון שאינכם יכולים לזייף את חתימות המתקנים. כמו כן, אינכם יכולים לשנות תוויות קיימות או להחליף חלקים מהן, כיוון שאינכם יכולים לחשב את סיכום הביקורת (אינכם יודעים את חתימת המתקן).

המודל הדמיוני הזה דומה מאוד לכרטיסי ה-VIP של דיסנילנד, רק שבדיסנילנד הוא יותר חולשתי... ☺

מתקפות נגד אימות מצביעים

בבסיסו, אימות מצביעים הוא אימות קריפטוגרפי לכל דבר. במקרים מסוימים, הוא עלול להיות פגיע למתקפות אימות קריפטוגרפיות קלאסיות.

מתקפות החלפה (Substitution attacks)

כפי שהוסבר בתיאוריית אימות מצביעים, באופן אידיאלי, לכל use case של מצביע צריכה להיות סכימת חתימה ייחודית לחלוטין המשויכת אליו. אך בפועל, סכימות חתימה חוזרות על עצמן ואינן תמיד ייחודיות (בשל מגבלות שונות ושיקולים מעשיים).

מתקפה זו מחליפה מצביע חתום שיועד ל-use case אחד במצביע חתום שיועד ל-use case אחר. אם שני המצביעים נחתמו באמצעות אותה סכימת חתימה, והסכימה אינה מגוונת מספיק, המצביע שנדרס עשוי לעבור אימות בהצלחה.

דוגמה פשוטה למתקפה זו היא החלפת מצביע חתום בטבלת פונקציות וירטואליות (vtable) במצביע חתום מ-vtable אחר. אם שני המצביעים נחתמו באמצעות אותה סכימת חתימה, ההחלפה עשויה להצליח.

אורקלים לחתימה והמרה (Signing/conversion oracles)

אורקל חתימה הוא קטע קוד שמקבל מצביע וחותם אותו באמצעות סכימת חתימה ספציפית, באופן שמאפשר לשלוף אותו מאוחר יותר (שומר את המצביע החתום לזיכרון / מחזיר אותו לקורא לפונקציה / וכו'). אורקלים כאלה יכולים להועיל לתוקפים, בהנחה שהם מסוגלים להוציא אותם לפועל.

אורקל המרה הוא קטע קוד שמקבל מצביע שחתום בסכימת חתימה מסוימת, מאמת אותו, וחותם אותו מחדש באמצעות סכימת חתימה אחרת, באופן שניתן לשחזור מאוחר יותר. אורקלים אלו יכולים להועיל לתוקפים, בהנחה שהם מסוגלים להריץ אותם כנדרש.

אורקלים לאימות (Authentication oracles)

אורקל אימות הוא קטע קוד שמקבל מצביע ובודק אם הוא חתום כראוי בסכימת חתימה ספציפית, ומדווח על התוצאה מבלי לגרום לקריסת התוכנית. ניתן להשתמש באורקלים כאלה כדי לבצע מתקפת bruteforce על ערך חוקי של תג למצביע.

דוגמאות לחולשות אמיתיות

PACMAN

PACMAN היא חולשה ברמת החומרה שמאפשרת לעקוף אימות מצביעים, שהתגלתה בשנת 2022 על ידי חוקרים מ-MIT. היא משפיעה על מעבדי ה-M1 של אפל (וייתכן שגם על שבבים אחרים המבוססים על ארכיטקטורת ARM).

מתקפת ה-PACMAN מוצאת דרך לגשת לאורקל אימות באמצעות מתקפת Side-Shannel ברמת החומרה. היא עושה זאת על ידי ניצול של "ביצוע ספקולטיבי" (Speculative Execution) - תכונה נפוצה לשיפור ביצועי המעבד, שבה המעבד מנחש אילו הוראות יגיעו בהמשך ומבצע אותן מראש.

לפרטים טכניים מעמיקים יותר על חולשת ה-PACMAN, ניתן לבקר באתר הרשמי המכיל קישור למאמר המקורי וסרטון מהרצאה בכנס DEFCON:

https://pacmanattack.com

CVE-2025-31201

CVE-2025-31201 הינה חולשה בספריית libRPAC.dylib של אפל (ספרייה קטנה המשמשת לבדיקות ביצועים ותהליכונים ב-Xcode ובמספר רכיבי מערכת) שאיפשרה לעקוף אימות מצביעים.

הספרייה ביצעה "Swizzling" למספר מתודות של Objective-C ושמרה את המצביעים המקוריים לפונקציות בזיכרון כתיב (אזור __DATA.__COMMON). המצביעים הללו נחתמו באמצעות המפתח IA, עם ערך הקשר 0 - בדיוק אותה סכימת חתימה ששומשה עבור Interposed Symbols (בחלק קריא בקובץ הבינארי). תוקף שכבר השיג יכולות קריאה וכתיבה יכול היה פשוט לדרוס את אחד מאותם מצביעים מקוריים שמורים עם כתובת של כל אחד מה Interposed Symbols (שכבר חתומים). כאשר המתודה שעברה Swizzling נקראה מאוחר יותר, הקריאה בעצם בוצעה על אותו Interposed Symbol במקום על המצביע במקורי. זוהי מתקפת החלפה.

Interposed Symbol שימושי כזה היה dlsym, שאיפשר לתוקף להשיג מצביעים חתומים לכל פונקציה מיוצאת (Exported function) בכל ספרייה שהיא.

לפרטים טכניים נוספים על חולשה זו, ניתן לקרוא כאן:

https://blog.epsilon-sec.com/cve-2025-31201-rpac.html#the-substitution-attack

https://www.wiz.io/vulnerability-database/cve/cve-2025-31201

שונות

חולשות ופגמים נוספים הקשורים לאימות מצביעים:

https://project-zero.issues.chromium.org/issues/42451026

https://project-zero.issues.chromium.org/issues/42451146

https://project-zero.issues.chromium.org/issues/42451144

סיכום

אימות מצביעים מספק מנגנון עוצמתי לאימות קריפטוגרפי של מצביעים בזמן ריצה, ובכך מעלה משמעותית את הרף עבור ניצול חולשות זיכרון המסתמכות על מניפולציה של מצביעים. אימות המצביעים מונע מתוקפים לזייף, להחליף או לשנות את ייעודם של מצביעים, אפילו בנוכחות יכולות של קריאה/כתיבה מהזיכרון.

ניתן לקרוא את המאמר אשר פורסם באנגלית:

https://idanrosenzweig.github.io/pointer_authentication_page/

אתגרי מחקר של אימות מצביעים

באתר researchlabs.tech, קיימים 4 אתגרים (בסגנון pwn) שבהם עליכם לעקוף את מנגנון אימות המצביעים:

- “next_gen_auth”
- “next_gen_auth_2”
- “next_gen_auth_3”
- “fun_land”

אודות המחבר

עידן רוזנצוייג, בן 19 מחיפה. עדיין מלשב, עובד בחברת סייבר כבר שנה וחצי. מתעסק במחקר ופיתוח בעולמות הסייבר. הקמתי לאחרונה אתר עם אתגרי מחקר בסגנון pwn, מוזמנים להיכנס ולעשות: researchlabs.tech. אני מכיר את המגזין כבר כמה שנים, מאוד אהבתי את הרמה הגבוהה והתכנים המגוונים והחלטתי לתרום גם בעצמי.

לכל שאלה הארה ופנייה מוזמנים לפנות אלי בכל מקום.

האימייל שלי: idanro12@gmail.com

הגיטהאב שלי: https://github.com/IdanRosenzweig

הלינקדאין שלי: https://www.linkedin.com/in/idan-rosenzweig-b82b1a1a5/

אתר המחקר שהקמתי: researchlabs.tech

מקורות מידע

- https://researchlabs.tech
- https://en.wikipedia.org/wiki/Return-oriented_programming
- https://developer.arm.com/documentation/102433/0200/Return-oriented-programming
- https://developer.arm.com/documentation/102433/0200/Jump-oriented-programming
- https://llvm.org/docs/PointerAuth.html
- https://clang.llvm.org/docs/PointerAuthentication.html
- https://projectzero.google/2019/02/examining-pointer-authentication-on.html
- https://github.com/pointer-authentication
- http://events17.linuxfoundation.org/sites/events/files/slides/slides_23.pdf
- https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/pointer-auth-v7.pdf
- https://www.usenix.org/system/files/sec19fall_liljestrand_prepub.pdf
- https://learn.arm.com/learning-paths/servers-and-cloud-computing/pac
- https://en.wikipedia.org/wiki/QARMA
- https://eprint.iacr.org/2016/444.pdf
- https://github.com/Phantom1003/QARMA64
- https://en.wikipedia.org/wiki/Setjmp.h
- https://en.wikipedia.org/wiki/Callback_(computer_programming)