Sequoia - CVE-2021-33909

אורי דאבוש ויוראי רוט חזן

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33909

הקדמה:

בתרגיל זה נתאר חולשה במנגנון ה-seq files של לינוקס.

seq files הם דרך שבה אנחנו מגדירים כיצד לפעול כאשר רוצים לפתוח קבצים מיוחדים כמו אלו הקיימים ב-proc/. היחוד של seq files זה שהם נועדו לעבוד עם טקסטים ארוכים יותר ובכך למנוע מהמתכנת לבצע גישה או שינוי לא נכונים - שיכולים לגרום לנזק רב.

במקרה שלנו, אנחנו משתמשים ב-seq files בשביל לגשת לקובץ $./proc/self/mountinfo$.

החולשה: (הסבר עם קוד)

ממשק קבצי ה-seq של הקרנל של לינוקס מיצר קבצים וירטואלים המכילים קטעים ורשומות.

כל רשומה חייבת להתאים לתוך באפר seq_file שמוארך בהתאמה על ידי הכפלת גודלו (שורה 242).

החולשה עצמה לא מגיעה מהכפלת הגודל בגלל ש m->size הוא מטיפוס size_t שמכיל 64 ביטים ולמערכת יגמר הזיכרון הרבה לפני שההכפלה תגרום ל-overflow. החולשה כאן היא בהעברתו כפרמטר לאחת הפונקציות שמצפה לקבל int ולא size_t (כלומר 32 ביטים במקום 64).

```
168 ssize_t seq_read_iter(struct kiocb *iocb, struct iov_iter *iter)
169 {
170         struct seq_file m = iocb->ki_filp->private_data;
…
205         /* grab buffer if we didn't have one */
206         if (!m->buf) {
207                 m->buf = seq_buf_alloc(m->size = PAGE_SIZE);
…
210         }
…
220         // get a non-empty record in the buffer
…
223         while (1) {
…
227                 err = m->op->show(m, p);
…
236                 if (!seq_has_overflowed(m)) // got it
237                         goto Fill;
238                 // need a bigger buffer
…
240                 kvfree(m->buf);
```

```
…
242                 m->buf = seq_buf_alloc(m->size <<= 1);
…
246         }
247 }
```

הערה: סימנתי בכחול איפה שהעבירו ונגעו בגודל. (או מה ששומר את הגודל)

ניתן לראות בקוד (מעל) שהגודל מועבר גם בשורה 227 שקוראת ל show_mountinfo שניתן לראות בקוד מתחת שקורא בשורה 150 ל seq_dentry, שקורא בשורה 530 ל dentry_path, שקורא בשורה 387 ל prepend.

```
135 static int show_mountinfo(struct seq_file *m, struct vfsmount *mnt)
136 {
…
150                 seq_dentry(m, mnt->mnt_root, " \t\n\");
-----------------------------------------------------------------------
523 int seq_dentry(struct seq_file *m, struct dentry *dentry, const char
*esc)
524 {
525         char *buf;
526         size_t size = seq_get_buf(m, &buf);
…
529         if (size) {
530                 char *p = dentry_path(dentry, buf, size);
-----------------------------------------------------------------------
380 char *dentry_path(struct dentry *dentry, char *buf, int buflen)
381 {
382         char *p = NULL;
…
385         if (d_unlinked(dentry)) {
386                 p = buf + buflen;
387                 if (prepend(&p, &buflen, "//deleted", 10) != 0)
-----------------------------------------------------------------------
11 static int prepend(char **buffer, int *buflen, const char *str, int
namelen)
12 {
13         *buflen -= namelen;
14         if (*buflen < 0)
15                 return -ENAMETOOLONG;
16         *buffer -= namelen;
17         memcpy(*buffer, str, namelen);
```

בפונקציה seq_read_iter בשורה 242 הבאפר של הGB2 מוקצה בזיכרון הוירטואלי בעזרת vmalloc ובשורה 227 show_mountinfo נקרא.

בפונקציה show_mountinfo בשורה 150 הפונקציה seq_dentry נקראת עם באפר ריק בגודל 2GB.

בפונקציה seq_dentry בשורה 530 הפונקציה dentry_path נקראת עם אורך ה-path המקורי (כלומר אורך הבאפר). אורך ה-path המקורי הוא 2GB כלומר 2^{32} והוא נשמר במשתנה מסוג size_t, ולאחר הקריאה לפונקציה והמרתו למשתנה מסוג int הערך שהוא מייצג הוא $2^{32} -$ (הערכים שיכולים להישמר במשתנה מסוג int הם בטווח $[-2^{32}, 2^{31} - 1]$ ולכן שמירת הערך 2^{32} גורמת לחריגה מגבולות ה-int ורק המידע שנמצא בגבולות ה-int נשמר).

בפונקציה dentry_path מוסיפים לבאפר את אורך הבאפר כדי לחשב את הסוף שלו, ובגלל שאורך הבאפר הפך לשלילי נקבל את הכתובת של התא 2GB לפני התחלת הבאפר. לאחר מכן, בשורה 387 הפונקציה prepend נקראת.

בפונקציה prepend בשורה 13 האורך של buflen מוקטן ב 10 והופך ל-$2^{31} - 10$ (underflow) הבאפר גם מוקטן ב 10 ומצביע ל 2GB-10B- מאחורי ההקצאה של vmalloc לבאפר ובשורה 17 ה 10 תווים "$//deleted$" נכתבים למה שאמור להיות 10 תווים לפני סוף הבאפר, אך בפועל נמצאים הרבה לפני כתובת הבאפר.

כעת נתאר כיצד ניתן להשמיש את החולשה הזו כדי לקבל הרשאות root:

1. יוצרים תיקייה שה-path אליה גדול מג'יגה בייט (זאת על ידי יצירת תיקייה בתוך תיקייה המון פעמים), עושים לה bind-mount ממשתמש ללא הרשאות גישה ומוחקים אותה.
2. יוצרים ת'רד שכותב תוכנית eBPF לזיכרון הוירטואלי (זיכרון הקרנל) בעזרת vmalloc וחוסמים אותו לפני שהתוכנית נכתבת לזיכרון הקרנל.
3. פותחים את הקובץ "$/proc/self/mountinfo$" ומתחילים לקרוא את ה-path הארוכה, מה שיגרום לכתיבת המחרוזת "$//deleted$" בהיסט של 2GB-10B- מתחילת הבאפר ש-vmalloc הקצתה.
4. מסדרים את התוכנית כך שהמחרוזת הנ"ל תדרוס פקודה (instruction) של תוכנית ה-eBPF שלנו (ותאפס את בדיקות האבטחה של ה-kernel eBPF verifier).
   כדי לסדר את התוכנית כך שהמחרוזת תדרוס את הפקודה של תוכנית ה-eBPF, אנחנו דואגים להקצות מקום ע"י שימוש ב-vmalloc לפני שיוצרים את התיקייה עם ה-path הארוך (ב-exploit המקורי הקצו 2 באפרים של 1GB לפני שהקצו את הבאפר עבור ה-path), כך נוכל לשחרר את הזיכרון המתאים כדי לדאוג שהזיכרון שיוקצה לתוכנית ה-eBPF יהיה בדיוק בהיסט המתאים כדי שהמחרוזת "$//deleted$" תיכתב איפה שאנחנו רוצים שהיא תיכתב. (אמור להיראות כמו בתמונה לאחר שסידרנו את ההקצאות זיכרון)

```
        "//deleted"
             |
      4KB    v   1GB        4KB        1GB        4KB        2GB
-----|---|---+-------------|---|-----------------|---|-----------------|
 ... |XXX| seq_file buffer |XXX| seq_file buffer |XXX| seq_file buffer |
-----|---|---+-------------|---|-----------------|---|-----------------|
         |   |                                     |
         |   \----<----<----<----<----<----<----<----/
        8182B                -2GB-10B
```

הערה: ניתן לראות איפה ה "$//deleted$" היה אמור להיכתב.

5. לאחר שהקצנו את כל הזיכרון עבור ה-path, אנחנו משחררים את הזיכרון המתאים (בעזרת vfree), כלומר זה שנמצא בהיסט אליו נכתוב בהמשך, ומשחררים את החסימה מהת'רד של תוכנית ה-eBPF כדי שהיא תיכתב לזיכרון בדיוק במקום בו אנחנו רוצים שהיא תיכתב.
נחסום את הת'רד שוב אחרי שהתוכנית אושרה ע"י ה-kernel eBPF validator (בנקודה זו יכתב ה"$//deleted$") ולפני שהיא קומפלה ע"י הקרנל.

```
        "//deleted"
             |
      4KB    v   1GB        4KB        1GB        4KB        2GB
-----|---|---+-------------|---|-----------------|---|-----------------|
 ... |XXX|  eBPF programs  |XXX| seq_file buffer |XXX| seq_file buffer |
-----|---|---+-------------|---|-----------------|---|-----------------|
         |   |                                     |
         |   \----<----<----<----<----<----<----<----/
        8182B                -2GB-10B
```

6. כעת אנחנו רוצים אפשרות לקרוא ולכתוב מידע למקומות שונים בזיכרון הקרנל. כדי לקרוא ולכתוב למקומות שונים בזיכרון הקרנל משתמשים בטכניקות שמנצלות חולשה בתוכניות eBPF (ניתן להרחיב כאן, זוהי חולשה נוספת לכן לא נכנסנו לעומקה).
7. בעזרת אפשרות הקריאה אנחנו מאתרים את הבאפר modprobe_path, ובעזרת אפשרות הכתיבה משנים את התוכן שלו (שהוא "$/sbin/modprobe$") ל-path ל-executable שלנו, שירוץ עם הרשאות root. כעת נקבל שברגע שנחזיר את הת'רד של תוכנית ה-eBPF שלנו התוכנית ששמנו את ה-path אליה תרוץ עם הרשאות root.

כיצד סגרו את החולשה?

כדי לפתור את הבעיה ולמנוע את החולשה, הוסיפו בדיקה לפונקציה seq_buf_alloc (שמקצה את המקום עבור ה-path) שתבדוק שהמקום שאנחנו מקצים לא עובר גבול מסוים.

הגבול נתון ע"י הקבוע MAX_RW_COUNT, שלפי הגדרתו לא עולה על INT_MAX.

```
if (unlikely(size > MAX_RW_COUNT))
        return NULL;
```

הגדרתו המקורית היא:

```
#define MAX_RW_COUNT (INT_MAX & PAGE_MASK)
```

ניתן לראות כאן את השינוי הנ"ל בקוד.