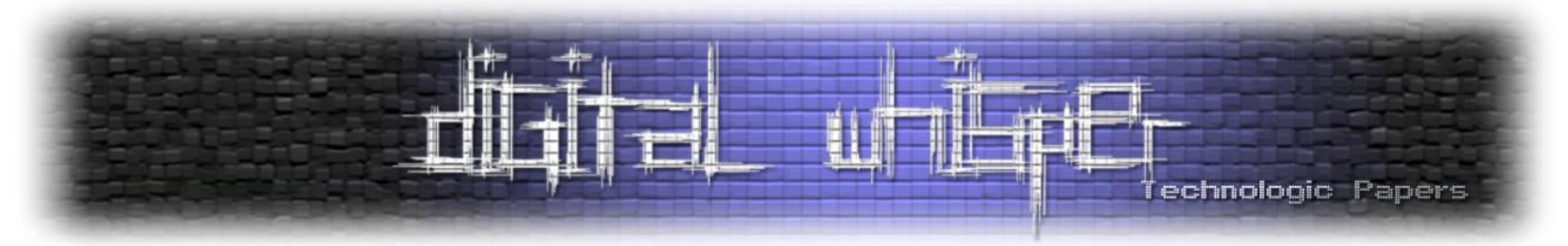


הרהורים על הממשק בין עולם המחקר ועולם הפיתוח FreeRDP כמקרה בוחן

מאת אייל איטקין

הקדמה

הסיפור מאחורי מאמר זה התחיל ב**ספטמבר 2018** במחקר חולשות שביצעתי כחבר בצוות המחקר של צ'ק-פוינט. המחקר נועד לבדוק וקטור תקיפה חדש לזמנו (Reverse RDP) עליו נרחיב במאמר זה. לשמחתנו, המחקר הוכיח את עצמו בזמן קצר יחסית ובאוקטובר אותה השנה התחלנו את תהליך דיווח החולשות לפרוייקטים הפגיעים: rdesktop, FreeRDP (פרויקטי קוד פתוח) ו-MSTSC של Microsoft.

המחקר עצמו כלל מספר סבבי חיפוש חולשות ופרסומים והיה אמור להגיע לסופו ב**יולי 2020** עם הפרסום של שרשרת הרצת קוד מלאה (כולל הדגמה) על Apache Guacamole המבוסס על FreeRDP. על פניו, סוף הסיפור, לא?

אז לא רק שלא זה המצב, מסתבר שכרונולוגית אפילו לא הגענו לחצי הדרך.

בעוד שמציאת חולשה, דיווח שלה והמתנה לתיקון מצד הפרויקט יכולים לקחת זמן, מדובר בסופו של יום בתיקון נקודתי של **חולשה אחת**. נהוג למדוד את התהליך בסדרי גודל של חודשים בודדים (תיקון תוך 90 יום) ולכן מדובר בתקופת המתנה לא נעימה, אך לא בדיוק פרק זמן ארוך במיוחד. בנוסף, ניתן לתהות כמה גדולה התועלת מתהליך נקודתי שכזה? הרי חולשות דומות כנראה ימשיכו להתגלות במוצר ויהפכו למשחק של חתול ועכבר בין המגנים לתוקפים.

במאמר זה נדון בניסיון לתקן בצורה **הרמטית** את בעיות הקוד ב-FreeRDP שגרמו לחולשות מלכתחילה, וציר הזמנים הכרוך בשינוי שכזה. מקרה הבוחן שלנו יהיה תיקון שהגשתי לפרויקט ב**אוקטובר 2021** ואשר בימים האחרונים (דצמבר 2023), בשעה טובה ומוצלחת נכלל רשמית בגרסה החדשה (3.0.0) של FreeRDP. קראתם נכון. התיקון נכנס לפרויקט לפני **שנתיים** והיה זמין בגרסת הפיתוח, אבל יושק בגרסה רשמית רק בתקופה הקרובה. במידה ותיקון זה היה מוכנס לפרויקט עוד ב-2018, הוא היה **חוסם קצת יותר ממחצית מחולשות הזלגת הזיכרון** שדווחו במוצר עד כה.

בעוד שבהתחלה התקשיתי להבין את ציר הזמנים הנ"ל בכובעי כחוקר חולשות, הרי שהיום לאחר מעבר לעולם הפיתוח אני יכול להעיד שלא מדובר על גרירת רגליים חלילה, אלא על תהליך הגיוני ותקין מצד מפתחי הפרויקט. במאמר זה אנסה להציג את FreeRDP מבעד לשתי נקודות מבט אלו (מחקר ופיתוח) בתקווה לספק הצצה לצד שזוכה להתייחסות מועטה למדי בעולם האבטחה: האנשים שמפתחים את הפרויקט עצמו.

Disclaimer

מאמר זה נכתב במקור בספטמבר 2023, טרום פרוץ מלחמת "חרבות ברזל". מעבר לעובדה שחלק מהנתונים שמוצגים בהמשך ככל הנראה השתנו מעט בזמן שחלף (למשל, סטטיסטיקה על חולשות ב-FreeRDP) הרי שהמאמר גם יועד לאתגר את "קהילת המחקר" בארץ ובחו"ל ואת יחסה לעולם הפיתוח.

המאמר נועד לקרוא לאחריות גדולה יותר של "קהילת המחקר" הישראלית והבינלאומית, ללא הפרדה ברורה בין השתיים. יותר שיתוף פעולה פנימי (בין חוקרים) וחיצוני (מול מפתחים), סטנדרטים גבוהים יותר ובאופן כללי בשלות ומקצועיות גבוהים יותר. בראייה לאחור, מדובר היה בפספוס כפול:

- ישנו הבדל משמעותי בין קהילת החוקרים הישראלית לזו הבינלאומית
- לא בטוח שהעיקרון של "גוף מקצועי בינלאומי" הוכיח את עצמו במבחן המציאות

איחוד השורות יוצא הדופן, כמו גם ההתגייסות הכללית של החברה הישראלית, הראו שללא צל של ספק אנחנו יודעים להירתם למשימה כשנדרש לכך. לעומת זאת, תגובות העולם לזוועות השביעי באוקטובר הראו, לצערנו הרב, שאל לנו לסמוך על גופים בינלאומיים שימלאו את תפקידם בשעת הצורך.

משכך, הרי שאין מה לתלות תקוות ב"קהילת מחקר" בינלאומית. קהילה שכזו אינה אפילו גוף רשמי אלא אוסף של פרטים וקבוצות המתהדרים בעיני עצמם בעבודה למען "עולם טוב יותר", לרוב ללא קשר ברור בין תואר זה לעבודתם בפועל.

ועדיין, לאחר לא מעט עדכוני ניסוח, בחרתי להשאיר במאמר את החלקים המתייחסים ל"עולם המחקר" אל מול "עולם הפיתוח". בעוד שאיני יכול לצפות מהעולם לשפר את דרכיו, כולנו יכולים, כל אחד ברמתו, לפעול לכך שלפחות בתחומים בהם אנחנו פועלים, נפעל כשאנחנו זוכרים ומתחשבים בנקודת המבט של האדם (מפתח או חוקר) שמולנו.

במאמר זה אנסה להמחיש איך הבנת העולם המקצועי של האחר תוביל, לפחות לדעתי האישית כחוקר לשעבר ומפתח בהווה, לשיפור משמעותי באבטחה של המוצרים השונים בהם אנו מתמקדים.

Remote Desktop Protocol & FreeRDP

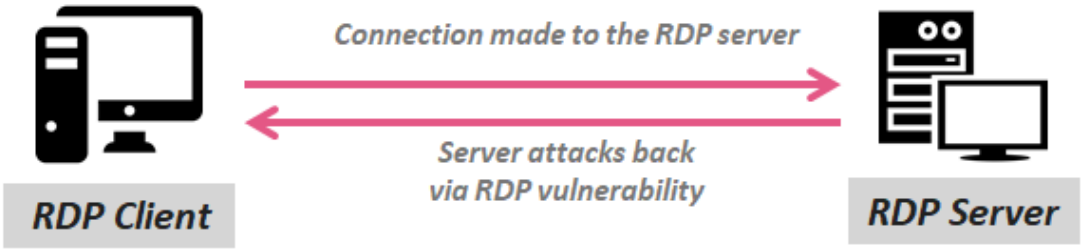
לפני שנצלול לווקטור התקיפה, כדאי שנתחיל מהסבר קצר על הפרוטוקול שבליבת מאמר זה - Remote Desktop Protocol או RDP בקצרה. הפרוטוקול נמצא בשימוש נפוץ על ידי משתמשים טכניים ואנשי IT אשר נעזרים בו כדי להתחבר למכונה מרוחקת. הפרוטוקול עצמו אופיין על ידי Microsoft ומוכר בעיקר סביב התוכנה mstsc.exe המובנית ב-Windows ומשמשת להתחברות אל מכונת Windows מרוחקת (בהתאם, פעולת התחברות זו זכתה לסלנג "למסטס").

בעוד שכמו מוצרים רבים של Microsoft, תוכנת הלקוח המדוברת הינה סגורה, הרי שישנם גם מספר פרויקטי קוד פתוח הממשים את הפרוטוקול ברמות בשלות כאלה ואחרות. מימושים שכאלה, בהם גם נתמקד בהמשך, כוללים את rdesktop הבסיסי יחסית (שנראה כי כבר אינו מתוחזק כיום) ואת FreeRDP המבוסס יותר. רק לתחושה, FreeRDP נפוץ מספיק שמספר לא קטן של מוצרי תקשורת מסחריים מבוססים עליו ואת חלקם נפגוש בהמשך. בנוסף, הפרויקט זכה להיכלל בתוכנית ה- Secure Open Source Rewards (sos.dev) אשר נועדה לספק תמריץ כלכלי לשיפור האבטחה של פרויקטי קוד פתוח שהוגדרו כ"קריטיים".

Reverse RDP

בתרחיש הנפוץ, משתמשים יעזרו בתוכנת לקוח של RDP על מנת להתחבר לשרת RDP מרוחק אשר רץ על המכונה אליה ירצו להתחבר. מכונה זו יכולה להיות מכונה וירטואלית, המכונה האישית שלנו במשרד כשאנחנו עובדים מהבית, או "סתם" מחשב שנמצא בעמדה לידנו ואין לנו כוח לקום עד אליו. מטרת ההתחברות היא לאפשר לנו גישה מלאה, בהרשאות המשתמש שלנו, על המכונה המרוחקת. כלומר, שליטה מלאה (הרצת קוד) על פי האפיון, זוהי מטרת הפרוטוקול.

אבל, מה יקרה אם תוקף יצליח להפוך את הכיוון? האם שרת RDP עוין יכול לנצל חולשה בפרוטוקול ה-RDP (שהינו מורכב למדי) על מנת להשיג שליטה דווקא על הלקוח התמים המנסה להתחבר אליו?



[**איור 1:** תיאור תרחיש התקיפה של Reverse RDP - לקוח מתחבר אל שרת עוין שבתורו מנצל את הפרוטוקול כדי לתקוף אותו בחזרה]

ישנם מספר תרחישים נפוצים בהם תקיפה שכזו יכולה לאפשר לתוקף לשדרג את מיקומו הרשתי בצורה משמעותית:

- תקיפת איש תמיכה טכנית המנסה להתחבר למכונה בארגון על מנת לפתור תקלה טכנית. תקיפה שכזו תאפשר לנו שליטה על עמדת מנהל רשת בעל הרשאות גבוהות בארגון.
- תקיפת חוקר Malware המנסה להתחבר למכונה וירטואלית המריצה נוזקה בסביבה שנועדה להיות מבודדת (Sandbox). תקיפה שכזו תוביל לבריחה מה-Sandbox.

ישנו גם תרחיש שלישי המאפשר למכונה זדונית ברשת להשתדרג עד לרמת גישה ארגונית כמעט מלאה, אבל על תרחיש זה נרחיב בהמשך.

התהליך הטבעי של המחקר היה בעקומת קושי עולה, כאשר התחלנו מללמוד על הפרוטוקול דרך פרויקטי הקוד הפתוח מהבסיסי (rdesktop) למורכב (FreeRDP), במטרה לשמור את mstsc לסוף. הנחת היסוד הייתה שלאורך הדרך נפתח הבנה של הפרוטוקול ושל טעויות נפוצות במימושו, כך שנגיע מוכנים יחסית לחיפוש חולשות במוצר הסגור של Microsoft אשר מהווה רוב מוחלט מנתח השוק של תוכנות הלקוח ל-RDP.

חולשות במימוש RDP

ואכן, ניכר היה שמדובר בווקטור תקיפה ממשי, לפחות בפרויקטי הקוד הפתוח מצאנו שלל חולשות די בסיסיות בטיפול בהודעות השרת. את החולשות חילקנו לשתי קבוצות עיקריות:

- קריאה מחוץ לזיכרון, במטרה להזליג מידע חזרה לתוקף (Info Leak).
- חולשות שיבוש זיכרון שיאפשרו כתיבה ואף השתלטות על הלקוח המתחבר (Remote Code Execution).

להלן שתי חולשות לדוגמא שמצאנו, אחת בכל קטגוריה:

rdesktop - CVE-2018-8798 - Info Leak:

```
  static void
  rdpsnd_process_training(STREAM in)
  {
          uint16 tick;
          uint16 packsize;
          STREAM out;
+         struct stream packet = *in;
+
+         if (!s_check_rem(in, 4))
+         {
+                 rdp_protocol_error("rdpsnd_process_training(), consume of training data from stream would overrun", &packet);
+         }

          in_uint16_le(in, tick);
          in_uint16_le(in, packsize);

          logger(Sound, Debug, "rdpsnd_process_training(), tick=0x%04x", (unsigned) tick);

          out = rdpsnd_init_packet(SNDC_TRAINING, 4);
          out_uint16_le(out, tick);
          out_uint16_le(out, packsize);
          s_mark_end(out);
          rdpsnd_send(out);
  }
```

[**איור 2**: התיקון של חולשת הזלגת הזיכרון CVE-2018-8798 ב-rdesktop - הוספת בדיקה כי בהודעה ישנם לפחות 4 בתים]

מדובר בחולשה מייצגת למדי של טיפול בקלט ללא בדיקה מקדימה כי הקלט הנ"ל בכלל נשלח על ידי השרת. למעשה, מדובר במוטיב חוזר שבו לפני **כל** טיפול בשדה בהודעה, על המפתחים לזכור לבדוק כי נשלחו מספיק בתים בכדי לכלול את השדה המדובר.

במקרה זה, מדובר בקריאה של 2 שדות בני 2 בתים כל אחד, אשר נקראים ומיד נשלחים בחזרה לשרת. כלומר, הזלגה של 4 בתי זיכרון הממוקמים מיד לאחר ההודעה שלנו, בתקווה שמדובר יהיה במצביע שיאפשר לנו ללמוד על מבנה הזיכרון של הנתקף.

:FreeRDP - CVE-2018-8786 - Heap-based Buffer Overflow

```
        if (Stream_GetRemainingLength(s) < 2)
                goto fail;

        Stream_Read_UINT16(s, bitmapUpdate->number); /* numberRectangles (2 bytes) */
        WLog_Print(update->log, WLOG_TRACE, "BitmapUpdate: %"PRIu32"", bitmapUpdate->number);

        if (bitmapUpdate->number > bitmapUpdate->count)
        {
-               UINT16 count;
-               BITMAP_DATA* newdata;
-               count = bitmapUpdate->number * 2;
-               newdata = (BITMAP_DATA*) realloc(bitmapUpdate->rectangles,
-                                                sizeof(BITMAP_DATA) * count);
+               UINT32 count = bitmapUpdate->number * 2;
+               BITMAP_DATA* newdata = (BITMAP_DATA*) realloc(bitmapUpdate->rectangles,
+                                        sizeof(BITMAP_DATA) * count);

                if (!newdata)
                        goto fail;

                bitmapUpdate->rectangles = newdata;
                ZeroMemory(&bitmapUpdate->rectangles[bitmapUpdate->count],
                           sizeof(BITMAP_DATA) * (count - bitmapUpdate->count));
                bitmapUpdate->count = count;
        }

        /* rectangles */
        for (i = 0; i < bitmapUpdate->number; i++)
        {
                if (!update_read_bitmap_data(update, s, &bitmapUpdate->rectangles[i]))
                        goto fail;
        }
```

[**איור 3**: התיקון של חולשת שיבוש הזיכרון CVE-2018-8786 ב-FreeRDP - שמירת ערך החישוב במשתנה מסוג UINT16 במקום משתנה מסוג UINT32]

גם כאן ניתן להבחין בכך שהטיפול בקלט חייב להתחיל בקריאה ל-()Stream_GetRemainingLength לפני שנקרא את כמות המלבנים בהם נצטרך לטפל.

עם זאת, החולשה הפעם נובעת מפעולת הכפל על הקלט ושמירת התוצאה במשתנה של UINT16 למרות שערך החישוב יכול להגיע לטווח של 17 ביטים. לצורך ההמחשה:

1. bitmapUpdate->count = 0x8001
2. bitmapUpdate->count * 2 = 0x10002
3. UINT16 count = 0x10002 & 0xFFFF = 2

מאוחר יותר, לולאת הטיפול במלבנים עצמם תתבצע לפי הערך הגבוה של 0x8001 בעוד שהקצאת הזיכרון נעשתה לפי ערך קטן יותר של 2, דבר שיוביל לדריסת זיכרון על ה-Heap.

סיכום החולשות שנמצאו

לאחר סבב המחקר הראשוני, החולשות שמצאנו הן:

- **rdesktop (19):** CVE 2018-8791, CVE 2018-8792, CVE 2018-8793, CVE 2018-8794, CVE 2018-8795, CVE 2018-8796, CVE 2018-8797, CVE 2018-8798, CVE 2018-8799, CVE 2018-8800, CVE 2018-20174, CVE 2018-20175, CVE 2018-20176, CVE 2018-20177, CVE 2018-20178, CVE 2018-20179, CVE 2018-20180, CVE 2018-20181, CVE 2018-20182
- **FreeRDP (6):** CVE-2018-8784, CVE 2018-8785, CVE 2018-8786, CVE 2018-8787, CVE 2018-8788, CVE 2018-8789
- **Mstsc (1):** No CVE

סה"כ 26 חולשות, על פני 3 מימושים שונים. החולשה שמצאנו ב-mstsc.exe מאפשרת לתוקף לנצל אירוע העתק-הדבק במהלך חיבור ה-RDP כדי להטיל קבצים נשלטים לנתיב נשלט במערכת הקבצים של המחשב המותקף. לדוגמא, יכולת זו יכולה לאפשר כתיבת Malware לתיקיית ה-Startup של המשתמש, דבר שיוביל להרצת קוד לאחר איתחול מחדש של המכונה. תקיפה זו הודגמה בסרטון הבא.

הכחשה, כעס, מיקוח, דיכאון, וקבלה

למרות תוצאות המחקר הראשוני, שלדעתנו היו משכנעות למדי, הופתענו לגלות תגובות צוננות מצד Microsoft בנוגע לחולשות שדיווחנו. ההתייחסות ההתחלתית **לווקטור התקיפה** הייתה כאילו הוא "לא מעניין", כפי שבאה לידי ביטוי בתגובה הבאה:

> *"Thank you for your submission. We determined your finding is valid but does not meet our bar for servicing. For more information, please see the Microsoft Security Servicing Criteria for Windows (https://aka.ms/windowscriteria)."*

או בתרגום חופשי לעברית:

> *"תודה לכם על ההגשה. החלטנו שהממצא שלכם נכון אך אינו עומד ברף שלנו לטיפול. למידע נוסף ראו...".*

עם זאת, כנראה שמאחורי הקלעים כן החל תהליך כלשהו, משום שבחודשים העוקבים חלו התפתחויות במספר צירים:

- עבודה משותפת שלנו עם חוקרת מ-Microsoft (דנה בריל) לקראת הצגה משותפת של צד התוקף וצד המגן שהתקבלה ל-Black Hat 2019 (קישור להרצאה בכנס).
- הבנה שלנו שהחולשה שמצאנו רלוונטית גם ל-Hyper V היות ונעשה שימוש ב-RDP מאחורי הקלעים עבור הממשק הגרפי המשמש לגישה אל המכונה (קישור לסרטון הדגמה).

- הפנמה של Microsoft שאכן מדובר בחולשה שדורשת התייחסות והקצאת מזהה החולשה CVE-2019-0887 (ומאוחר יותר CVE-2020-0655 בשל תיקון חלקי של החולשה).
- התחלת עבודה של Microsoft על מחקר חולשות עצמאי בנושא, ולבסוף הצגת הממצאים (13 חולשות) + חשיבות וקטור התקיפה (גם עבור Windows Sandbox) במסגרת Blue Hat IL 2020 כאן בכנס שלהם בארץ (קישור להרצאה שלהם בכנס).

אחת החולשות שנמצאה על ידי Microsoft תוקנה באוגוסט 2019 ונכללה במשפחת החולשות שזכתה לכינוי DejaBlue. בחינה של התיקון הראתה שמדובר היה בווריאנט של חולשה שעזרנו לתקן ב-FreeRDP כ-10 חודשים קודם לכן:

```
@@ -377,8 +408,15 @@ int zgfx_decompress(ZGFX_CONTEXT* zgfx, const BYTE* pSrcData, UINT32 SrcSize, BY
                        if (!zgfx_decompress_segment(zgfx, stream, segmentSize))
                                goto fail;

+                       if (zgfx->OutputCount > UINT32_MAX - used)
+                               goto fail;
+
+                       if (used + zgfx->OutputCount > uncompressedSize)
+                               goto fail;
+
                        CopyMemory(pConcatenated, zgfx->OutputBuffer, zgfx->OutputCount);
                        pConcatenated += zgfx->OutputCount;
+                       used += zgfx->OutputCount;
                }
        }
```

[**איור 4:** התיקון של CVE-2018-8785 שמצאנו ב-FreeRDP (2018) - הוספת בדיקה כנגד Integer Overflow]

```
InputOffset = *(_DWORD *)&input_buffer[_InputOffset] + segmentDataStartOffset;
ConcatenatedOffset += OutputCount;
if ( (unsigned int)OutputCount + ConcatenatedOffset < ConcatenatedOffset )// Integer-Overflow check
  return 0x80070216;
if ( ConcatenatedOffset > *(_DWORD *)(input_buffer + 3) )// numBytesWritten + OutputCount > uncompressedSize ==> error
  break;
memcpy_0((void *)(*((_QWORD *)this + 8) + ConcatenatedOffset), OutputBuffer, (unsigned int)OutputCount);
if ( ++segmentNumber >= (unsigned int)*(unsigned __int16 *)(input_buffer + 1) )// segmentNumber < segmentCount
  goto LABEL_22;
```

[**איור 5:** התיקון של CVE-2019-1181/2 (DejaBlue) שנמצאו על ידי Microsoft ב- mstsc (2019) - הוספת בדיקה כנגד Integer Overflow]

עצירת ביניים - השקפה על הממשק בין עולם המחקר ועולם הפיתוח

בניגוד לפרויקט הממוצע, ב-Microsoft יש לנו שלוש נקודות מבט:

- **חברת פיתוח גדולה:** גוף פיתוח גדול עם דעה בסיסית זו או אחרת על "הפרעות" מצד חוקרי אבטחה.
- **חברה בעלת גוף מחקר אבטחתי:** על פניו גוף המחקר הפנימי מוכוון לשפר את יכולת החברה לנתח איומים ולהתמודד איתם. בנוסף, תפקידו להקל על הממשק מול חוקרי אבטחה חיצוניים (כמונו).
- **חברה מובילת שוק:** חברת Microsoft היא מענקיות התוכנה בעולם וכבר לא מעט שנים שהיא משקיעה משאבים משמעותיים גם בגוף האבטחה הפנימי, תוך מאמץ למצב את החברה כמובילת שוק גם בתחום האבטחה. עדות ברורה לכך היא ההשקעה בכנסים השונים של BlueHat ברחבי העולם, כמו גם איכות המחקרים המוצגים בכנסים אלו, בין אם על ידי חוקרים חיצוניים ובין אם על ידי עובדי החברה.

במצב רגיל, הייתי מניח שהתגובה הראשונית שקיבלנו היא עדות נוספת לתחלואות של ארגון גדול (מעל מאתיים אלף עובדים) - קושי לשכנע מנהלים, קיבעון מחשבתי וכו', ומסיים כאן. במצב שכזה, דווקא לפרויקטי קוד פתוח קטנים יותר יש יותר סיכוי להיות בעלי קשב לדיווחים מצד חוקרים.

כאילו כדי לחזק טענה זו, לאחר כמה חודשים גוף המחקר (הבינלאומי) הראה שהוא כן משוכנע בחשיבות וקטור התקיפה, ערך מחקר המשך פנימי בעצמו ואף הציג זאת בכנס המחקר של החברה (BlueHat IL). כל הסימנים המעידים על בעיית תקשורת בתוך ארגון גדול - גוף המחקר לא הצליח לשכנע את צד הפיתוח של הארגון.

במעבר על שתי הנקודות הראשונות, חברת פיתוח גדולה וקיום גוף אבטחה פנימי, נראה שהחברה נהנתה מקיומו של גוף אבטחה (שחסר בגופי פיתוח קטנים יותר), אבל במקרה זה הבירוקרטיה הכללית של חברה גדולה היא זו שהכריעה את הכף (דבר שאולי יפעל לטובת גופים קטנים יותר).

אבל מה בנוגע לנקודה השלישית? לפחות חלק מגוף האבטחה של Microsoft היה מודע לתהודת המחקר שפרסמנו (במסגרת זה הרי הגיע מחקר המשך משותף שהוצג ב-Black Hat) ומשום כך היה מודע גם לחשיפה ולביקורת סביב התגובה האנמית שהתקבלה מצד החברה בנוגע לחוסר חשיבות ווקטור התקיפה. האם הניסיון למצב את החברה כמובילת שוק בעולם האבטחה עזר לתקן את הרושם הראשוני הבעייתי שהתקבל כתוצאה מתגובת גוף הפיתוח בחברה?

לצערנו, גם כאן התשובה היא **לא**. לא רק שצעדי המחקר של Microsoft נעשו בצורה **עצמאית** וללא שום יצירת קשר איתנו, גם האזכור בנוגע לחשיבות וקטור התקיפה הוצנע עמוק בדקה 34 של ההרצאה, ושימש בסה"כ כרקע לתיאור מחקר התקפי פנימי (red team) שביצעה החברה.

לולא היינו בכנס עצמו אפילו לא היינו יודעים כי החברה השתכנעה שאכן מדובר בווקטור תקיפה הדורש התייחסות. בנוסף, באי הכנס שלא הכירו את המחקר שלנו עוד היו עלולים להתרשם שמדובר על וקטור תקיפה חדש שהוצג בכנס לראשונה, היות ולא הייתה שום התייחסות או הפנייה למחקרים קודמים שנעשו

בתחום. בהינתן הנוהג המקובל בעולם המחקר להפנות לעבודות קודמות בצורה ברורה (כדי לתת הקשר ורקע, נוהג אשר נפוץ לא רק באבטחה אלא במחקר בכלל), מדובר בהתנהלות מפתיעה.

בראייה מערכתית, דעתי היא שדווקא הנקודה האחרונה היא החשובה ביותר. הסיבה העיקרית לכך, והיא תוזכר שוב בהמשך, היא ש**הדבר החשוב ביותר שצריך להבין על גופי פיתוח הוא שהם תמיד עמוסים**. תמיד יש עוד גרסה, עוד לקוח ועוד דברים לתקן. על כן, צורת ההתייחסות הכללית לדברים מבוססת על תעדוף: "כמה גדול הלקוח שמחכה לגרסה? כמה הוא יכעס אם היא תתעכב?", "כמה תועלת תצמח מזמן העבודה על התכולה החדשה? האם עדיף להשקיע את המשאבים במקום אחר?".

בעולם שכזה, הממשק בין עולם הפיתוח לעולם האבטחה, מבוסס על פרמטרים לתעדוף:

- **חקיקה/תקינה:** מוצרי אבטחה לשיפור תהליך הפיתוח (סריקת מוצרים לחולשות) יוטמעו כאשר הרגולציה תחייב את התקנתם.
- **כללי אצבע / רשימת חולשות נפוצות:** כפי שמוזכר באתר של OWASP Top 10 (לאחר תרגום): "מוכר על ידי מפתחים ברחבי העולם בתור הצעד הראשון לקראת פיתוח תוכנה אבטחתי יותר".

אם אנחנו, כקהילת מחקר אבטחתי, לא נצליח להגיע להסכמה רחבה בנוגע לאיומים / בעיות קוד הדורשים התייחסות, אנחנו לא יכולים לצפות מעולם הפיתוח שיצליח לעקוב בעצמו אחרי העדכונים בתחום. ללא עדכון שכזה, היכולת לתעדוף אפקטיבי היא מוגבלת ולכן תשומת הלב והמשאבים ינותבו למשימות אבטחתיות ברמה פחותה (במקרה הטוב) או למשימות פיתוח אחרות (במקרה הרע).

בניגוד לחוויית הדיווח ל-Microsoft, כאשר פנינו לפרויקטים הקטנים יותר, הם ידעו שהשעון מתקתק כי יש להם 90 יום לתקן את הבעיה + לשחרר את התיקון. הם כולם הגיבו לעניין, בצורה מקצועית, ושחררו תיקונים מהירים ומוצלחים לבעיות שדיווחנו להם. אבל, ויש כאן אבל גדול, הטיפול בבעיות שדווחו היה במתכונת של טיפול במשבר, תחת לחץ ובצורה נקודתית.

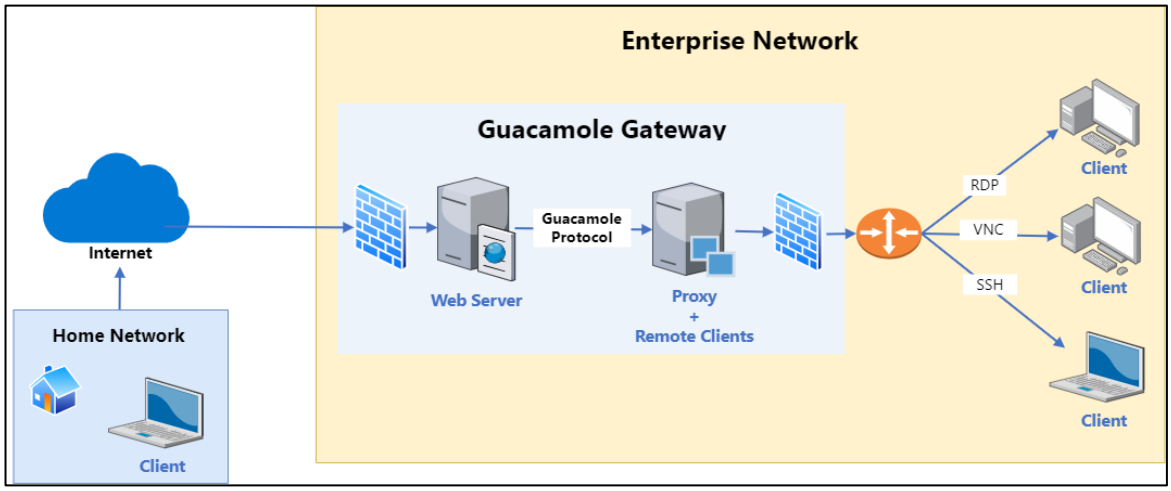
האם אנחנו מצפים שצוות פיתוח בודד יצליח להסיק מהממצאים שדיווחנו להם כי יש גם בעיה רוחבית בקוד שלהם שצריכה לקבל מענה? כאמור, הם רואים רק את מה שאנחנו דיווחנו להם והם לא חשופים לחולשות דומות שמצאנו במתחרים שלהם. האם אנחנו מצפים שהם יגדירו לעצמם משימות המשך, ברמת תיעדוף גבוהה, כדי להמשיך לתקן את שורשי הבעיות בקוד שלהם שהובילו לבעיות הנקודתיות במוצר? אחרי הכל, לגופים הללו אין צוות אבטחה פנימי שיעזור בתעדוף, בתכנון דרך התיקון הנכונה או בתהליך הבקרה על איכות התיקון.

בסופו של יום, גם אם יתבצע תהליך הפקת לקחים כלשהו מצד צוות הפיתוח, הצוות יאלץ לעשות זאת לבד, בלי סיוע מאיתנו. אנחנו הרי מזמן חזרנו לדרכנו ופרסמנו בלוג זה או אחר אודות המחקר שעשינו. בזאת תם תפקידנו, לא?

דעתי האישית, ועל כך ארחיב בהמשך, היא שבמקרה הטוב הציפיות שלנו מצוותי הפיתוח הן לא ריאליות. במקרה הרע, אין לנו בכלל ציפיות כאלה מהם וגם לנו אין אינטרס ברור שיהיה תיקון רוחבי במוצר כי לא זו מטרת המחקר שלנו.

הצצה לשורש הבעיה ב-Apache Guacamole - FreeRDP כמקרה בוחן

עם תחילת סגר הקורונה בשנת 2020, קיבלנו פרויקט חדש למחקר, המוצר Apache Guacamole. המוצר, המבוסס על FreeRDP, מהווה פתרון נפוץ לגישה מרוחקת אל המחשב הארגוני בזמן העבודה מהבית. להלן איור הממחיש את ארכיטקטורת הרשת הרצויה בעת שימוש במוצר:



[**איור 6:** תיאור ארכיטקטורת הרשת בעת השימוש ב-Apache Guacamole בכניסה לרשת הארגונית]

כפי שניתן להבין מהציור, המוצר מהווה מעין Proxy שמצד אחד מתפקד כשרת התחברות (אימות משתמשים) ומהצד השני נעזר בלקוח RDP, VNC או SSH כדי להתחבר אל המחשב הארגוני (בהתאם להגדרות שהוגדרו לשרת).

המשמעות היא שבעת ההתחברות של עובד אל המחשב הארגוני שלו, תוקף שישב על המחשב הארגוני יוכל לתקוף בחזרה (Reverse RDP) את שרת ה-Proxy של Guacamole. בתרחיש קיצון, תקיפה שכזו תוביל לכך שכל עובד שיתחבר אל המכונה הארגונית שלו למעשה יאפשר לתוקף לשלוט בצורה מלאה בשרת ה-Proxy דרכו עוברים **כל** החיבורים של עובדי הארגון. בצורה שכזו תוקף יוכל להשתדרג מעמדה רגילה ברשת (עובד יחיד), לגישה אל כל המכונות בארגון (כל העובדים). שליטה של תוקף ביתר החיבורים היא דוגמא קלאסית של מתקפת "אדם שבאמצע" (Man-In-the-Middle).

ואכן, במהלך המחקר מצאנו מספר חולשות אשר אפשרו בנייה של שרשרת תקיפה מלאה הממחישה בדיוק את התרחיש הנ"ל - השתלטות על שרת ה-Proxy ומשם שליטה מלאה על החיבור של יתר העובדים בחברה. להלן קישור לסרטון ההדגמה.

התקיפה שבהדגמה נעזרת בשתיים מהחולשות שמצאנו:

- CVE-2020-9497 - הזלגת זיכרון נשלטת לפי שדה אורך של 16-ביט (בדיוק כמו heartbleed).
- CVE-2020-9498 - מצביע רפאים (Dangling Pointer) המאפשר שיבוש זיכרון (כתיבה) ואפשר לנו להגיע עד לכדי הרצת קוד.

החולשה הראשונה שימשה להזלגת כתובות זיכרון במטרה לשבור ASLR וללמוד על מבנה זיכרון המכונה הנתקפת ואילו החולשה השנייה שימשה להשתלטות על המכונה עצמה. לפרטים נוספים אודות המחקר מוזמנים לעיין בבלוג המלא אודות המחקר.

בעיות יסודיות בעיצוב התוכנה של FreeRDP

במהלך המחקר הבחנו בשתי בעיות עיקריות בעיצוב התוכנה של FreeRDP:

- **תכנון מועד לתקלות:** לקראת **כל** טיפול בשדה מהודעה נכנסת, על המפתח לזכור לבדוק האם שדה זה בכלל התקבל כדי שלא יהיה מצב שאנחנו קוראים מעבר לגבולות ההודעה.

```
if (Stream_GetRemainingLength(s) < 2)
{
    WLog_ERR(TAG, "Stream short");
    return FALSE;
}
Stream_Read_UINT16(s, numberOrders); /* numberOrders (2 bytes) */
```

[**איור 7:** שימוש ב-()Stream_GetRemainingLength לבדיקה כי בהודעה ישנם לפחות 2 בתים, לפני שהשדה numberOrders (באורך 2 בתים) יקרא מגוף ההודעה]

- **קו הגנה יחיד:** פונקציית הבדיקה הוגדרה לפי ערך חזרה של size_t שהינו אי-שלילי (unsigned) במקום טיפוס המסוגל להכיל ערכים שליליים (signed). לכן, **במידה וקראנו מעבר לגבול ההודעה**, המשמעות תהיה ש**כל יתר בדיקות הקלט יעברו** ללא קשר לאורך ההודעה.

לדוגמא, קריאה של 2 בתים מחוץ לגבולות ההודעה תוביל ל"אורך הודעה נותר" של $2^{64}-2$ בתים, מה שבפועל יוביל לכך שכל יתר הבדיקות מסוג "האם אפשר לקרוא X בתים?" יעברו:

```
static INLINE size_t Stream_GetRemainingLength(wStream* _s)
{
    return (_s->length - (_s->pointer - _s->buffer));
}
```

[**איור 8:** מימוש הפונקציה ()Stream_GetRemainingLength המציג כי ערך החזרה הינו size_t (אי-שלילי)]

השילוב של שתי בעיות העיצוב הללו הוא אחת הסיבות העיקריות לריבוי החולשות שנמצאות לאורך השנים ב-FreeRDP ובמוצרים המבוססים עליו.

בנוסף, גם אם נכשיר את כל מפתחי הפרויקט הנוכחיים להיות מודעים לחשיבות בדיקות הקלט, עוד מעבר לטעויות אנוש שתמיד יכולות לקרות, קשה להאמין שהידע הזה יעבור באופן מושלם לכל מפתח חדש

שיתרום קוד לפרויקט. אחרי הכל, מדובר במוצר תוכנה התומך בהרחבות, ובמקרה של Apache Guacamole, התקיפה התבססה על חולשות בהרחבה אותה מימש הפרויקט עצמו.

מצב זה, בו אבטחת המוצר מבוססת על כך שכל המפתחים יזכרו כי עליהם למקם בדיקות קלט בכל פיסת קוד שיוסיפו למוצר / להרחבה שלו, דן את המוצר להיות פגיע אבטחתית. בהינתן עיצוב פגיע זה, ובהינתן תשומת הלב המוגבלת של צוות הפיתוח שרואה כל עץ בנפרד בבואו לתקן חולשה שדווחה לו, אך לא רואה את היער במלואו, איך פותרים את הבעיה בצורה רוחבית?

לעניות דעתי, התשובה חייבת להגיע מגורם חיצוני. ברוב המקרים, מצידם של חוקרי אבטחה להם יש את ההבנה האבטחתית אודות מהות הבעיה. ההנחה הגלומה בנקודת מבט זו היא כי לחוקרים ישנו רצון לעבוד **יחד** עם המפתחים של הפרויקט כדי לחזק משמעותית את אבטחת המוצר.

תהליך התיקונים ב-FreeRDP

כאמור, הבעיה אותה נרצה לפתור היא היכולת של תוקף לקרוא מחוץ לגבולות ההודעה, בדגש על שימוש ביכולת זו כדי להזליג מידע זיכרון. אחת הסיבות לדגש זה היא שללא זליגת זיכרון, היכולת של תוקף לעקוף ASLR ו/או מנגנוני הגנה אחרים ב-Heap הינה מוגבלת. בצורה זו אנחנו יכולים לבנות שכבות על גבי שכבות של הגנה, כאשר כל שכבה מטפלת בבעיה אחת פשוטה ומוגדרת.

כדי לטפל בבעיה בצורה רוחבית, עלינו להתייחס לשתי בעיות היסוד שמצאנו בעיצוב התוכנה ואותן תיארנו קודם לכן. ההתייחסות יכולה להיות תיקון (העלמת הבעיה) או הורדת התמריץ של תוקף לנצל את הבעיה (גביית מחיר).

תיקון פונקציית בדיקת הקלט

על פניו, מדובר בתיקון הכי פשוט בעולם, החלפה של size_t ב-ssize_t בחתימת פונקציית הבדיקה. בסה"כ הוספה של תו יחיד (+ תיקון שגיאות הקומפילציה הנובעות מהמרת הסוגים) והעלמנו את הבעיה:

```
-        static INLINE size_t Stream_GetRemainingLength(wStream* _s)
+        static INLINE ssize_t Stream_GetRemainingLength(wStream* _s)
```

[**איור 9:** עדכון חתימת הפונקציה Stream_GetRemainingLength() לכדי ערך חזרה ssize_t (יכול להכיל ערכים שליליים)]

שינוי זה יבטיח שגם אם פוספסה בדיקה בפונקציה כלשהיא, בדיקה עתידית בהמשך המסלול תתפוס את הבעיה ותפיל את ההודעה.

הערה חשובה: בעוד שנראה שמדובר בשינוי קטן (תו בודד), הרי שמבחינת עולם התוכנה מדובר דווקא בשינוי גדול למדי. כפי שראינו במקרה של Apache Guacamole, הפונקציה הנ"ל היא חלק מה-API ש-FreeRDP מספקת למפתחי תוכנה לצורך מימוש הרחבות. כלומר, **מדובר בשינוי API**.

השינוי יכול להיראות לנו זניח, וסביר שהוא לא ישבור את הקוד של משתמשים, אבל מנקודת המבט של עולם הפיתוח, זוהי שבירה של ה-API. משום כך, היא דורשת התייחסות בהתאם על מנת להימנע מפגיעה בחוויית השימוש של הלקוחות. לדוגמא, קימפול מחדש של תוספים שהשתמשו ישירות ב-type ערך החזרה וכעת צריכים לבצע עדכונים מתאימים גם אצלם בקוד.

בדיקה מרוכזת של תקינות מעטפת ההודעה

אחת הבעיות בעיצוב הנוכחי היא הצורך למקם בדיקות קלט לפני כל גישה לשדה מהודעה נכנסת. ברור לכל שגישה זו אינה חסינה וכי יהיו מקומות בקוד בהם ישכחו למקם בדיקות קלט ("חזקה כי קו המגע לעולם ייפרץ"). טעויות שכאלו יוכלו להיות עוד בהכנסת הקוד במימוש הראשוני של תכולה זו או אחרת, או בעדכון התכולה לאורך הזמן.

על כן, במקום להעלים את הבעיה, נוכל לנסות לתפוס אותה בצורה שתעניש תוקף שינסה לנצל את החולשה. כל הודעה נכנסת עוברת טיפול בהתאם לתוכן שלה, על פני אינספור מסלולי קוד אפשריים. אבל, בסופו של המסלול, ההודעה תמיד תגיע למקום אחד ויחיד - שחרור ההודעה בחזרה למאגר ההודעות.

אם נוסיף לפונקציית השחרור בדיקה שתבדוק חריגה אפשרית של ראש הקריאה מהגבולות החוקיים של ההודעה, הרי שנוכל לפעול על בסיס אירוע חריג זה:

```
@@ -253,8 +255,11 @@ void StreamPool_Return(wStreamPool* pool, wStream* s)
        StreamPool_Lock(pool);

        StreamPool_EnsureCapacity(pool, 1, FALSE);
-       pool->aArray[(pool->aSize)++] = s;
-       StreamPool_RemoveUsed(pool, s);
+       {
+               Stream_EnsureValidity(s);
+               pool->aArray[(pool->aSize)++] = s;
+               StreamPool_RemoveUsed(pool, s);
+       }

        StreamPool_Unlock(pool);
}
```

[**איור 10:** עדכון הפונקציה StreamPool_Return() בה מוחזרת ההודעה למאגר ההודעות, כך שההודעה תיבדק לתקינות טרם החזרתה]

פונקציית הבדיקה עצמה בסה"כ צריכה לבדוק הנחות יסוד בסיסיות על תקינות ההודעה:

- ראש הקריאה/כתיבה (s->pointer) חייב להיות בכתובת גדולה או שווה מתחילת ההודעה.
- כמות הבתים שקראנו חייבת להיות קטנה או שווה לכמות הבתים בהודעה.
- כמות הבתים שכתבנו חייבת להיות קטנה או שווה לכמות הבתים שהוקצתה להודעה.

```
void Stream_EnsureValidity(wStream* s)
{
        size_t cur;

        STREAM_ASSERT(s);
        STREAM_ASSERT(s->pointer >= s->buffer);

        cur = (size_t)(s->pointer - s->buffer);
        STREAM_ASSERT(cur <= s->capacity);
        STREAM_ASSERT(s->length <= s->capacity);

        /* Length is only valid after a call to Stream_SealLength */
        if (s->length > 0)
        {
                STREAM_ASSERT(cur <= s->length);
        }
}
```

[**איור 11:** הפונקציה שהוספנו במטרה לבדוק את תקינות מבנה ההודעה]

במקרה שאחד מהתנאים מופר, המאקרו STREAM_ASSERT() יבצע רישום ללוג ולאחר מכן יוביל לסגירת התהליך:

```
#define STREAM_ASSERT(cond)
   do
   {
       if (!(cond))
       {
           const char* tag = "com.freerdp.winpr.wStream";
           WLog_FATAL(tag, "%s [%s:%s:%" PRIuz "]", #cond, __FILE__, __FUNCTION__, __LINE__);
           winpr_log_backtrace(tag, WLOG_FATAL, 20);
           abort();
       }
   } while (0)
```

[**איור 12:** רישום ללוג וסגירת התהליך במידה ותנאי הבדיקה אינו מתקיים]

מהות השינוי היא כי התקפה דו-שלבית (הזלגה ואז שיבוש זיכרון) אינה אפשרית יותר היות והתהליך יצא כבר אחרי שלב ההזלגה, עוד בטרם נספיק להיעזר במידע זה עבור שלב התקיפה הבא.

מבחינת תוקף, ההשפעה של שינוי זה מחולקת לשני מקרים הדורשים התייחסות:

1. **הזלגת זיכרון למטרת עקיפת ASLR או התגברות על מנגנוני הגנה אחרים:** במקרים שכאלו, סגירת התהליך סותמת את הגולל על ניסיון תקיפה עתידי באמצעות המידע שנאסף. זאת כמובן תחת הנחת היסוד כי לכל תהליך ישנו טווח כתובות המוגרל מחדש עבורו בעלייה (ASLR).
2. **הזלגת סוד כללי של המערכת:** במידה וקיימת, תוקף עדיין יוכל לנצל חולשת הזלגת זיכרון המאפשרת הזלגת סוד כללי של המערכת. במידה ולסוד זה השפעה רחבה על המערכת, עוד מעבר לתהליך הנתקף שזה אך נסגר, הרי שהתוקף עדיין השיג נכס בעל ערך. יחד עם זאת, תוקף ישקול האם שווה להסתכן ברישום ללוג + הפלת התהליך כולו. אחרי הכל, תוקף תמיד יעדיף להימנע מחשיפה ומדובר בשתי פעולות רועשות למדי.

הערה: בנוגע לתרחיש #1, יש לציין שלא זה המצב בארכיטקטורה של Apache Guacamole, שם התבצעה בחירה מעניינת ליצור כל תהליך מחדש באמצעות fork() מהתהליך הראשי ואז קריאה לפונקציה, במקום קריאה ל-execve(). בחירה בעייתית זו הייתה חלק חשוב בתרחיש התקיפה המלא שהדגמנו על המערכת.

משמעויות התיקון

בעוד שמדובר על שינויים קטנים יחסית בקוד, טיב השינויים והמיקום שלהם מאפשרים התמודדות רוחבית עם בעיות עיצוב התוכנה שב-FreeRDP. במידה ותיקונים אלו היו קיימים בפרויקט מלכתחילה, הם היו חוסמים את החולשות הבאות / משנמכים אותן ל-Denial-of-Service (DoS) בלבד:

- **FreeRDP (23)**: CVE-2018-8789, CVE-2020-4032, CVE-2020-11018, CVE-2020-11040, CVE-2020-11042, CVE-2020-11043, CVE-2020-11045, CVE-2020-11046, CVE-2020-11047, CVE-2020-11048, CVE-2020-11049, CVE-2020-11058, CVE-2020-11085, CVE-2020-11086, CVE-2020-11087, CVE-2020-11088, CVE-2020-11089, CVE-2020-11099, CVE-2020-11526, CVE-2020-13396, CVE-2022-39316, CVE-2022-39319, CVE-2023-39350
- **Apache Guacamole (3*)**: CVE-2020-9497

23 מתוך 81 מהחולשות שפורסמו ב-FreeRDP כחלק מהמחקר שלנו או אחריו, היו נחסמות או כבר נחסמו על ידי התיקון שהכנסנו למוצר. מדובר על **כ-28% מהחולשות במוצר!** והתיקון לקח בסה"כ יומיים עבודה.

בנוסף, ניזכר שאנחנו מנסים לטפל רק בחולשות המאפשרות הזלגת זיכרון (45) ולא בכל החולשות (81). על כן החישוב הוא 23 מתוך 45 - קצת יותר **ממחצית (51%) מחולשות הזלגת הזיכרון היו נחסמות**.

הערות נוספות:

- Apache Guacamole הקצו מזהה חולשה 1 ל-3 חולשות הזלגת זיכרון שמצאנו במוצר שלהם.
- הדו"ח של החוקרים שדיווחו על CVE-2020-11526 מציין גם הוא את הבעיה בחתימה של פונקציית בדיקת הקלט.

- כבר בתיאור החולשה CVE-2023-39350 ניתן לראות כי הם מציינים שמנגנון ההגנה שהוספנו ל-FreeRDP מצמצם את החולשה מהזלגת זיכרון ל-DoS בלבד.

ניתוח נוסף של החולשות ב-FreeRDP על פני השנים

ניתוח נוסף של הנתונים מראה מספר מגמות מעניינות בנוגע לחולשות שדווחו ב-FreeRDP:

- במוצר דווחו 91 חולשות, כאשר החולשה הראשונה דווחה בשנת 2013.
- בשנים 2013-2017 דווחו במוצר **10 חולשות**.
- בשנת 2018 דווחו ממצאי המחקר שלנו שכלל 6 חולשות, ולאחריהן דווחה חולשה אחת נוספת.
- בשנים 2019-2023 דווחו במוצר **74 חולשות** - גידול משמעותי:
  - 2019 - 2
  - **2020 - 40**
  - 2021 - 4
  - 2022 - 11
  - 2023 - 17
- כאמור, 23 מתוך 81 מהחולשות שפורסמו ב-FreeRDP כחלק מהמחקר שלנו או אחריו, היו נחסמות או כבר נחסמו על ידי התיקון שהכנסנו למוצר.

בנוסף, קשה לבודד משתנים בנוגע לסיבה בגידול דיווחי החולשות במוצר לאחר פרסום המחקר שלנו (תחילת 2019) אבל סביר להניח שהמרכיב המשמעותי היה המעבר לשימוש מוגבר במוצרים מבוססי FreeRDP בעקבות הקורונה. הסיבה לקושי בבידוד המשתנים היא כי באותה התקופה היו גם ריבוי מחקרים אודות חשיבות וקטור התקיפה Reverse RDP.

בעוד שלא ניתן להעריך בצורה מדויקת אם הסיבה העיקרית לקפיצה היא תקופת הקורונה או העניין המחקרי טרום-קורונה, כן ניתן לראות אילו חולשות דווחו במוצר. מעבר על רשימת החולשות מראה **שרובן המוחלט הוא וריאנטים של המחקרים שהתפרסמו בנושא עוד לפני הקורונה**. ממש ברמת דיווחים נוספים על אותם קבצים/פונקציות בהן דווחו חולשות קודמות או חולשות ב-mstsc.exe.

למעשה, הנתונים הללו מעידים כי **בעולם אבטחת המידע ישנה העברת ידע מוצלחת למדי**. צוותי מחקר וחוקרים עצמאיים ברחבי העולם (לפחות לפי השפות בדיווחי החולשות לפרויקט) מתבססים על מחקרים קודמים ונעזרים בהם במחקרי ההמשך שלהם.

מדובר בחדשות די משמחות בנוגע לאפקטיביות המחקרית בפרסום מחקרי חולשות.

ציר הזמנים לתיקון

בתחילת המאמר הזכרתי כי תהליך התיקונים לקח קצת מעל **שנתיים**, אבל חשוב להבין איך הגענו למצב הזה. כפי שניתן לראות מה-pull request שריכז את תיאור הבעיה, העקרונות המנחים לפתרון, הצעת התיקון שלי והשיח עם המפתחים, התהליך כולו לקח פחות מ**יומיים**. אז מה הוביל לשחרור מאוחר כל כך של התיקון לציבור?

כאן נכנס לתמונה ציר הזמנים של עולם פיתוח התוכנה. בראש ובראשונה, תיקון התוכנה נכנס לענף הפיתוח הראשי של הפרויקט. משם, במידת הצורך, התיקון יועבר גם לענפי הגרסאות הפעילים של המוצר. גם אם היינו מכניסים את התיקון לגרסה רשמית ונתמכת של המוצר, הרי שהיינו צריכים לחכות לשחרור גרסה חדשה שתכלול את התיקונים.

כאשר מדובר בתיקון חולשה, הרי שנצפה לתיקון תוך 90 יום דבר שיכריח את צוות הפיתוח לוודא שתשוחרר גרסאת תוכנה חדשה בפרק זמן זה עבור כל דורות המוצר שנמצאים בתחזוקה פעילה. במקרה זה, לא מדובר בתיקון חולשה, ועל כן טבעי לצפות שהתיקון יכנס לרשימת השינויים שתשוחרר לציבור כאשר תהיה גרסה חדשה של המוצר.

אבל, כאן התהליך מסתבך. היקף השינויים שעשינו בקוד, לרבות שינוי חתימת פונקציה, גורמים לכך שמדובר על **שינוי API** של המוצר. אם שינוי זה ישוחרר בגרסה עתידית, הוא יוביל לשבירת API של לקוחות קיימים המשתמשים במוצר (binary compatibility) ו/או בונים את המוצר שלהם מעליו (source compatibility). אז מתי אפשר לעשות שינוי שכזה?

עולם התוכנה התכנס ברובו למתכונת הבאה:

- גרסאות פרויקטים יסומנו באמצעות X.Y.Z:
  - X - גרסה ראשית
  - Y - גרסה משנית
  - Z - גרסאת תיקונים (patch version)
- שיטה זו נקראת Semantic Versioning והיא עוזרת להגדיר את טיב שינויי התוכנה בכל סוג גרסה:
  - עדכון גרסה ראשית - מותר לשבור API
  - עדכון גרסה משנית - הוספת תכולות ללא שינוי ב-API
  - עדכון גרסאת תיקונים - תיקוני קוד קלים ללא תכולות חדשות (מכאן שמה)

כלומר, אם במהלך הוספת התיקון שינינו את ה-API, הרי שעלינו לחכות לשחרור **הגרסה הראשית הבאה** של הפרויקט. במקרה שלנו, הגרסה הנוכחית היא 2.Y.Z ועלינו לחכות ל-3.0.0.

ברובם המוחלט של פרויקטי התוכנה הרציניים, שינוי גרסה ראשית הוא לא דבר של מה בכך. בהגדרה הוא משדר ללקוחות שאולי שברנו להם את ה-API, ואנחנו עלולים להבריח לקוחות. בנוסף, אנחנו רוצים

להתחייב ללקוחות על פרק זמן ארוך כלשהו (נגיד, שנתיים או שלוש) של תמיכה פעילה במוצר. לכן, אם כל חצי שנה נוציא גרסה ראשית חדשה, אנחנו עלולים למצוא את עצמנו תומכים במספר רב של גרסאות במקביל, על כל העלויות המשתמעות מכך.

דוגמא בולטת לכך היא פרק הזמן שלקח ל-OpenSSL לעבור מגרסה 1.X (מרץ 2010) לגרסה 3.X (ספטמבר 2021). כן, הם דילגו מעל המספר 2 מסיבה לא ברורה. כלומר, עבור פרויקט כל כך גדול, שנמצא בשימוש נרחב בכמות כמעט אינסופית של פרויקטים, פרק הזמן הדרוש היה **מעל עשור**.

אם נסתכל חזרה על FreeRDP, כנראה שפרק זמן של שנתיים הוא לא כזה נורא. מה גם שאלו שנתיים מתוך כ-5 שנים שחלפו בין הוצאת גרסה 2.0.0 וגרסה 3.0.0.

מה זה אומר? זה אומר שיש יתרון לתיקוני קוד נקודתיים, כי הם יגיעו לכל גרסאות המוצר שנמצאות בשימוש ובפרק זמן קצר יחסית (חודשים בודדים). אבל, זה גם אומר ששינויי קוד גדולים יותר דורשים תכנון מראש וניהול קפדני. במידת האפשר, יש להקפיד שלא לשבור את ה-API של המוצר, על מנת לאפשר לשינוי להגיע ללקוחות מהר יותר (בסדרי גודל).

אבל, גם תקופה של שנתיים תגיע בסוף, ועל כן רצוי לעבוד עם מפתחי הפרויקט על השינויים כדי שיום מן הימים יהיה תיקון רוחבי במקום אוסף של פלסטרים נקודתיים. אחרי הכל, תיקון רוחבי (אך קטן) שסוגר 28% מהחולשות הוא כן משהו ששווה לכוון אליו, גם אם הוא לוקח זמן.

ומה עושים ברגע שהגרסה הראשית הנוכחית השתחררה? מתחילים לעבוד על שינויים גדולים לגרסה הבאה בתור. ככה עולם הפיתוח עובד, מדובר בריצה למרחקים ארוכים.

רשמים מתהליך התיקונים

זו הפעם השנייה בה אני מנסה להכניס מנגנון אבטחה רוחבי בפרויקט תוכנה גדול, ולא רק עוזר להם בתיקון נקודתי של בעיה אבטחתית זו או אחרת. המקרה הקודם היה עבודה משותפת שלי עם מפתחי glibc על מנת להכניס מנגנון הגנה בשם Safe-Linking ל-Heap. הרשמים שלי לאחר העבודה המשותפת עם שני פרויקטים אלו, מעידים המון דברים טובים על עולם פיתוח התוכנה ולצערי מאירים באור הרבה פחות חיובי את עולם המחקר.

זה כן המקום לסייג שהיו גופים, חברות לרוב, אשר תהליך דיווח החולשות אליהם היה לא נעים ונע על הטווח שבין חוסר עניין מצידם לעוינות של ממש. אין לי שום כוונה לטעון שעולם הפיתוח (או כל גוף אחר) הינו מושלם, אבל הניסיון שלי הוא שברגע שעושים את המעבר מדיווח על חולשה (90 יום וכל הטקס) להושטת יד במטרה לתקן בעיה במוצר (ללא CVE), זוכים לאוזן קשבת בתהליך.

שיתוף פעולה

בשני המקרים המפתחים הגיבו הרבה יותר טוב משציפיתי ותוך זמן קצר מאוד עזרו להכניס את התיקון לפרויקט. פעם אחת (glibc) אני שלחתי את תיקון הקוד ובפעם השנייה (FreeRDP) התיקון הראשוני שלי היווה בסיס לתיקון שלם יותר שנעשה על ידי המפתחים עצמם תוך התייעצות איתי.

רצון לשפר את אבטחת הפרויקט

בשני המקרים ניכר היה מהשיח עם צוותי הפיתוח כי אבטחת הפרויקט חשובה להם, גם אם היא לא זוכה לתיעדוף בים המשימות השוטפות של הפרויקט. כלומר, בהינתן אירוע חיצוני שמסב את תשומת לבם לשיפור בעלות-תועלת טובה יחסית, הרי שהם ישמחו להכניס את השיפור ויפגינו מוטיבציה גבוהה במהלך התהליך. כפי שהזכרנו בעצירת הביניים, מילת המפתח כאן היא **תעדוף,** וביכולתנו להטות את הכף כדי שהתעדוף יהיה בעד הכנסת שיפור אבטחתי במוצר.

יתרה מכך, במקרה של glibc, התיקון הראשוני נכנס בגרסה 2.32 והתהליך הוביל להתעניינות מצידם במחקרי אבטחה נוספים אודות טכניקות תקיפה על malloc וה-heap כולל מעבר שלהם על מאמרים מאתגרי CTF שונים העוסקים ב-heap. בסופו של התהליך, שיפורים נוספים הוכנסו לה-heap לרבות ביטולם של ה-malloc hooks למיניהם לאחר חישוב עלות-תועלת עצמאי של צוות הפיתוח והחלטה שלהם להסירם בגרסה 2.34.

מדובר בתוצאות משמעותית יותר טובות ממה שניתן היה להעלות על הדעת בנקודת ההתחלה.

מיעוט שיפורים אבטחתיים

אם אנסה לספור כמה פעמים ראיתי פרסום של חוקרי אבטחה אודות שיפור אבטחה רוחבי שהם עזרו להכניס במוצר זה או אחר (תרגיל מחשבתי לקוראים), הרי שיישארו לי אצבעות עודפות בתהליך. ואפילו מתוך פרסומים אלו, רובם יהיו של צוות אבטחה הגנתי (blue team) המתאר את השיפורים שהם עזרו להכניס במוצרי החברה בה הם עובדים.

יש שיטענו כי מיעוט של שיפורים אבטחתיים הוא תוצאה ישירה של המורכבות הטכנית שלהם. אולם, אם נסתכל על Safe-Linking בתור דוגמא, הרי שלא מדובר על המצאה מהפכנית. מדובר על מיסוך המצביע ברשימה חד-כיוונית (XOR עם ערך אקראי) כדי שלא ניתן יהיה לדרוס את המצביע ולהפנות את הרשימה

לכתובת הנשלטת על ידי תוקף. עדות נוספות לפשטות מנגנון זה ניתן למצוא ב- Chrome, אשר בצורה בלתי תלויה הכניסו מנגנון הגנה כמעט זהה לתוך מימוש ה-Heap שלהם עוד בשנת 2012.

רק לשם השוואה, בשנת 2005 כבר הוכנס Safe-Unlinking לרשימות הדו-כיווניות. אולם, למרות פשטות המנגנון של Safe-Linking, הרשימות החד-כיווניות ב-glibc זכו למענה אבטחתי רק בשנת 2020. אפילו כשפותח מנגנון זה ב-Chrome, לא רק שהוא לא הופץ הלאה למימושים אחרים בתעשייה, הוא אפילו לא מצא את דרכו אל gperftools של Google עצמה, ממנו הגיע במקור מימוש ה-Heap של Chrome.

אין לי ספק כי שיפורים אבטחתיים כן דורשים נקודת מבט אבטחתית, גם אם הם פשוטים בעיקרם. עם זאת, האם ישנו הסבר למיעוט השיפורים בהשוואה לכמות חוקרי האבטחה או לכמות המחקרים ההתקפיים?

מחקר הגנתי מול מחקר התקפי

האם הציפיות שלנו מחוקרי אבטחה "עצמאיים" לא כוללות שיפור אבטחת מוצרים מעבר לתיקון של עוד חולשה זו או אחרת? לתשובה לשאלה זו יש השלכה ישירה על גורלם של פרויקטי קוד פתוח להם אין יכולת להעסיק צוות אבטחה הגנתי בכדי שיעזור לשפר את המוצר שלהם.

אם ננסה לאמץ את זיכרוננו, כאשר ישנם פרסומים הגנתיים של חוקרים "עצמאיים", איזו במה קיבלו הפרסומים? ואילו תגובות? האם אנחנו באמת מתמרצים חוקרים לעזור להגן על העולם? או שאולי אנחנו רק מוכרים את זה כסיפור כדי להרגיש טוב עם עצמנו בזמן שפעם בכמה זמן אנחנו "נופלים" על צוות פיתוח מסכן? אותו צוות פיתוח פתאום צריך לתקן חולשה (באילוצי זמנים לא קלים) והכל כדי שאנחנו נוכל להציג לכל העולם את הטעות שמצאנו בקוד שלהם.

המוטיבציה למחקר והמחיר של פרסום

לאורך תפקידי כחוקר אבטחה בצוות שעוסק ברובו במחקר למטרות פרסום, השתדלתי לחקור נושאים שעניינו אותי אישית תוך ניסיון לאזן בין עניין, מקצועיות ועמידה בציפיות המעסיק. ועדיין, בין אם מדובר היה בפרסום כלי מחקר, הצגת כיוון מחקר חדש (חולשות או Malware) או הצגת מנגנון הגנה/שיפור, הפוקוס אמור להיות על המטרה שהוגדרה למחקר. מטרה זו היא לעולם לא השמצת הפרויקט הנחקר.

יחד עם זאת, תמיד ישנו מתח טבעי בין האדרת תוצרי המחקר (מצג שווא של איום גדול יותר מזה שמצאנו) והימנעות מהשמצת הפרויקט בו נמצאה החולשה ("אנחנו" המוצלחים ו"הם" שלא מבינים מהחיים שלהם). אני גם אודה שלא הרגשתי בנוח עם 100% מהפרסומים של המחקרים שלי, בייחוד לאור העובדה שהם

עברו שרשרת ארוכה שכללה אנשי שיווק ועיתונאים עד שהגיעו לציבור, ולפעמים היה נראה שמדובר על משחק גדול של "טלפון שבור".

הנטייה הטבעית של כל אחת מהחוליות בשרשרת הפרסום תמיד תהיה העצמת הסיפור. זהו מתח תמידי שעל החוקר להילחם בו כדי לשמור על מקצועיות מחד ולשמור על כבוד הפרויקט בו נמצאה החולשה מאידך. **חוקר שיגיע לסיטואציה הזו תוך התמקדות חלילה ביוקרה ובמוניטין של עצמו, תוך התעלמות מצד הפרויקט שבסיפור, עלול בקלות להתפתות להתנשא מעל צוות הפיתוח, דבר שלא יועיל ליחסים תקינים בין הצדדים.**

כל אלו לא בדיוק תורמים לביסוס מערכת יחסית בריאה בין חוקרים ומפתחים.

הרושם האישי שלי לאחר קצת פחות מ-3.5 שנות מחקר חולשות למטרות פרסום היה ששני המחקרים בהם השפעתי הכי הרבה על עולם האבטחה היו בכלל מחקר שקשור לעולם ה-Malware (אותו פרסמנו כאן ב**גיליון 123**) והשיפור ב-glibc, כאשר האחרון זכה לתגובות צוננות ("קשה למכור לציבור מחקר כזה") עד עוינות ("למה שתרצה לשפר את האבטחה ב-malloc?"). סוג התגובות הראשון היה צפוי, באמת קשה למכור מחקר שכזה לציבור כאשר המטרה היא מחקר למטרות פרסום.

ועדיין, התגובות מהסוג השני היו לי יותר קשות, כי הן הגיעו מחוקרים. חלקן היו מבודחות ("זה יקשה על תרגילי CTF") אבל חלקן באמת הגיעו מחוקרים שרואים רק את הצד התוקף, כאילו הוא היחידי שחשוב, ואז למה שארצה לפגוע בו?

סיכום - המפתחים הם חלק מהתמונה

באנלוגיית המגן-תוקף בה פתחתי את המאמר הזכרתי משחק של חתול ועכבר בניסיון לרדוף אחרי חולשות נקודתיות בקוד. אבל הצגה זו היא מעט מעוותת. הרי המגן הוא חוקר אבטחה שמתיימר לעזור לעולם (באמצעות מציאת חולשה נקודתית ודיווח שלה) והתוקף הוא לרוב אותו החוקר בדיוק שינצל את החולשה הנ"ל כדי להתגאות ב-CVE חדש ובלוג/הצגה בכנס.

בעוד שקיימים כמובן מקרים של תוקפים "אמיתיים" שינצלו את החולשה לתקיפה למטרות רווח/נזק, הרי שלא איתם אנחנו משחקים חתול ועכבר. בכמה מקרים בשנים האחרונות התגלו חולשות בפרויקטי קוד-פתוח מפורסמים שנוצלו in-the-wild על ידי גופי פשיעה, ורק לאחר זיהוי התקיפה הן דווחו לפרויקט ותוקנו? כמעט לא זכורים לי כאלה. כמובן שיתכן שחוקרים יגרמו לתיקון חולשה שכבר נמצאת בשימוש של גופי תקיפה, אבל לרוב לא נקבל מידע על כך ולכן ואין לנו יכולת לדעת מה אפקטיביות ההגנה בה אנחנו מתגאים בהכריזנו על מציאת ותיקון חולשה נקודתית.

ובעצם, אנחנו שוכחים שישנו צד נוסף במשוואה והוא צוות הפיתוח של המוצר עצמו. מתי בפעם האחרונה ספרנו אותם בתור הצד ה"מגן" בסיפור? האם יש צידוק לזה שאנחנו מתייחסים אליהם כאל שחקן פאסיבי שרק מכניס חולשות לקוד אחת לכמה זמן?

לפחות מהחוויה האישית שלי, של שיתוף פעולה מוצלח בתהליך התיקונים, ניכר כי צוותי הפיתוח דווקא בעלי מוטיבציה לשפר את המוצר עליו הם עובדים. לעומתם, נראה שהמוטיבציה חסרה דווקא בצד שלנו של הסיפור. אנחנו בסה"כ זורקים עליהם חולשות נקודתיות פעם בכמה זמן, פותחים להם שעון כדי למדוד כמה זמן לוקח להם לתקן ואז עוד מציגים לכל העולם את הפאדיחות שהם עשו.

לאור כל זאת, דעתי היא שכדאי שנתחיל להתייחס לצוות הפיתוח כשחקן בתחום וננסה לשתף פעולה במטרה להיות **יחד** המגן בסיפור - דרך עבודה עמוקה ורוחבית יותר, ולא רק הצבעה על בעיות נקודתיות.

הפניות למידע נוסף

במהלך מאמר זה נעזרתי בתוכן ואיורים ממאמרים קודמים שכתבתי במסגרת קבוצת המחקר של צ'ק פוינט. להעמקה נוספת, אתם יותר ממוזמנים לעיין במאמרים הבאים:

- Reverse RDP - https://research.checkpoint.com/2019/reverse-rdp-attack-code-execution-on-rdp-clients/
- Reverse RDP - Hyper V Connection - https://research.checkpoint.com/2019/reverse-rdp-the-hyper-v-connection/
- Reverse RDP - The Path Not Taken - https://research.checkpoint.com/2020/reverse-rdp-the-path-not-taken/
- Would you like some RCE with Your Guacamole? - https://research.checkpoint.com/2020/apache-guacamole-rce/