WinRAR - un’esposizione lunga 19 anni

Author : Daniele Rigitano

Date : 14 Marzo 2019

In questo articolo riportiamo nel dettaglio il modo in cui i ricercatori di *CheckPoint* sono venuti a conoscenza di una vulnerabilità, presente nelle dll utilizzate da WinRAR, che permette di ottenere il controllo completo di un dispositivo[1].

WinRAR è un’utilità di archiviazione di file “trialware”, che può manipolare archivi RAR/ZIP e decomprimere numerose altre tipologie di archivi.

Secondo lo stesso produttore di WinRAR, oltre 500mln di utenti rendono WinRAR lo strumento di compressione oggi più popolare al mondo.

L’exploit, attivato tramite la semplice estrazione di un archivio, mette in grave pericolo l’intera clientela che utilizza questa utilità. Tale vulnerabilità, vecchia di oltre 19 anni, ha obbligato gli sviluppatori di WinRAR a prendere in tempi brevi una decisione drastica per porre rimedio al problema.

SCENARIO DI ANALISI

CheckPoint ha creato di recente un laboratorio di analisi volto ad analizzare la sicurezza dei software di terze parti. Tale laboratorio usa tecniche innovative tra le quali il “*binary fuzzing*”, ovvero una tipologia di attacco brute-force al codice binario che, “bombardando” un software con input formati da dati casuali, permette di individuare la presenza di vulnerabilità all’interno del codice.

Lo strumento utilizzato per questo tipo di attività è una particolare versione di WinAFL, una fork di
AFL[2] - American Fuzzy Lop, creata da GoogleProjectZERO.

L’analisi è stata incentrata verso le *dll* ritenute obsolete, ma ancora gestite dell’applicazione.

Come è facile immaginare, già dai primi test il *fuzzer* ha prodotto risultati che hanno evidenziato

comportamenti “anomali” da parte di una *dll*. Tali anomalie hanno poi confermato la presenza di un bug logico denominato “*Absolute Path Traversal*”[3].

FASI PRELIMINARI DEL PROCESSO DI FUZZING

Prima partire con le analisi, sono richieste alcune operazioni preliminari volte ad ottimizzare l’esecuzione del fuzzer, tra le quali:

1. il *patching* dell’eseguibile WinRAR per:

a) processare qualsiasi tipo di archivio, senza dover ricorrere a test-case specifici per ogni formato;

b) disabilitare la GUI, evitando qualsiasi interazione umana;

2. reperire in rete della documentazione sulle utilità di archiviazione, come la ricerca condotta nel 2005 dall’Università di Oulu.

3. integrare WinAFL con le opzioni da riga di comando di WinRAR.

Fatto ciò è stato possibile procedere con il *fuzzing* dell’eseguibile.

I PRIMI TEST

I primi tentativi evidenziato delle vulnerabilità di tipo *Memory Corruption - Writes Out-of-Bounds* per gli archivi:

- RAR
- LZH
- ACE

Purtroppo sfruttare queste vulnerabilità è risultato da subito infruttuoso, poiché le primitive di estrazione offrono controlli limitati verso la manipolazione di buffer sovrascritti.

Malgrado ciò, un crash relativo al parsing del formato ACE ha attirato l’attenzione dei ricercatori.

WinRAR utilizza una *dll* chiamata *unacev2.dll* per manipolare questo tipo di archivi. Si tratta di una vecchia libreria, compilata nel 2006, senza alcun meccanismo di protezione. Ed è proprio questa libreria l’input della seguente analisi.

COSTRUIRE UN TEST-CASE SPECIFICO

Per analizzare la libreria è necessario capire come quest’ultima viene usata. Il reverse enginering del codice che invoca degli archivi ACE, ha evidenziato due funzioni chiave:

```
INT __stdcall ACEInitDll(pACEInitDllStruc DllData);

INT __stdcall ACEExtract(LPSTR ArchiveName, pACEExtractStruc Extract);
```

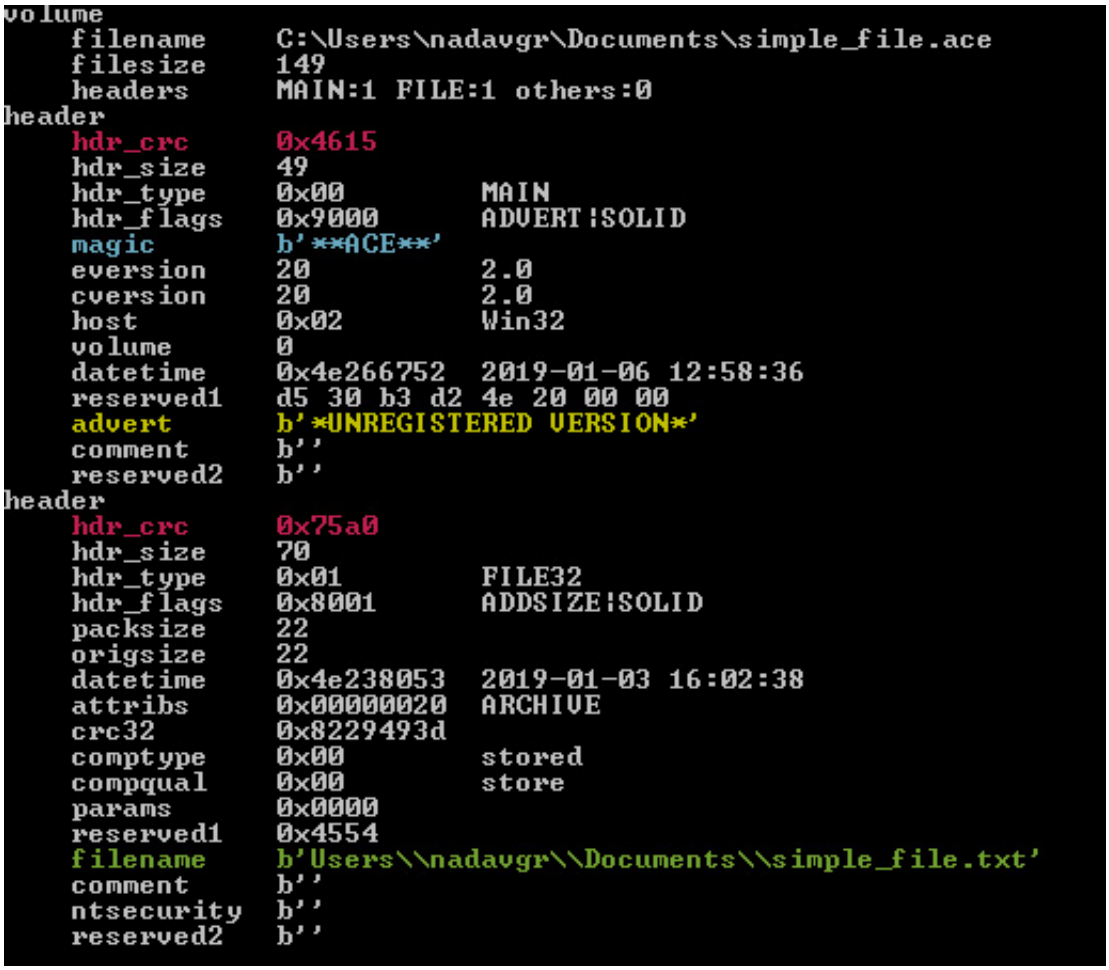
Le intestazioni dettagliate delle strutture dati in firma sono state ricavate da un progetto open source chiamato FarManager[4]. Tali informazioni hanno permesso l'affinamento dei vari test-case, per una migliore integrazione con il fuzzer.

COMPRENDERE IL FORMATO ACE

Non esiste un RFC per questo formato, ma vi sono numerose informazioni su internet a tal proposito:

- la creazione di un archivio ACE è protetta da un brevetto;
- l'unico software che è autorizzato a creare un archivio ACE è WinACE;
- l'ultima versione di questo programma è datata novembre 2007;
- il sito Web dell'azienda non è raggiungibile dell'agosto 2017;
- l'estrazione di un archivio ACE non è protetta da un brevetto.

Per comprendere a pieno il formato del file ACE è stato creato un archivio, col "vecchio" WinACE, contenente un semplice file ".TXT". Successivamente è stata utilizzata l'utility acefile per analizzare l'header di tale archivio.



I campi importanti sono:

hdr_crc (contrassegnato in rosa):

- campo di controllo integrità: se il CRC non corrisponde ai dati, l'estrazione viene interrotta.

filename (evidenziato in verde):

- contiene il percorso relativo al file. Tutte le directory specificate nel percorso relativo vengono create durante il processo di estrazione (incluso il file).

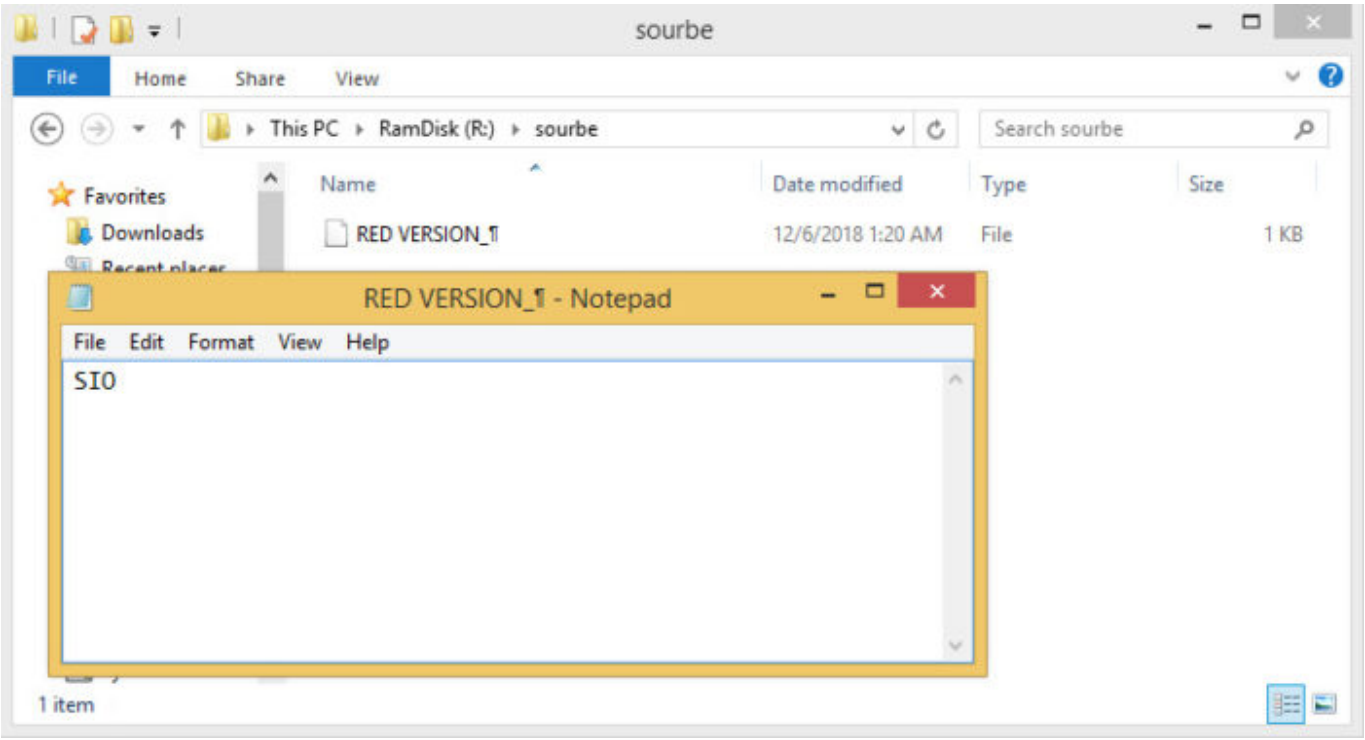
Poiché il campo *filename* contiene il percorso relativo al file, si possono sicuramente effettuare dei tentativi per verificare se l'algoritmo è vulnerabile al "*Path Traversal*".

Va sottolineato che, per quanto accennato in precedenza, per ogni tentativo di attacco vi è la necessità di generare un valore CRC valido da inserire nell'intestazione del file, per permettere l'estrazione dell'archivio.

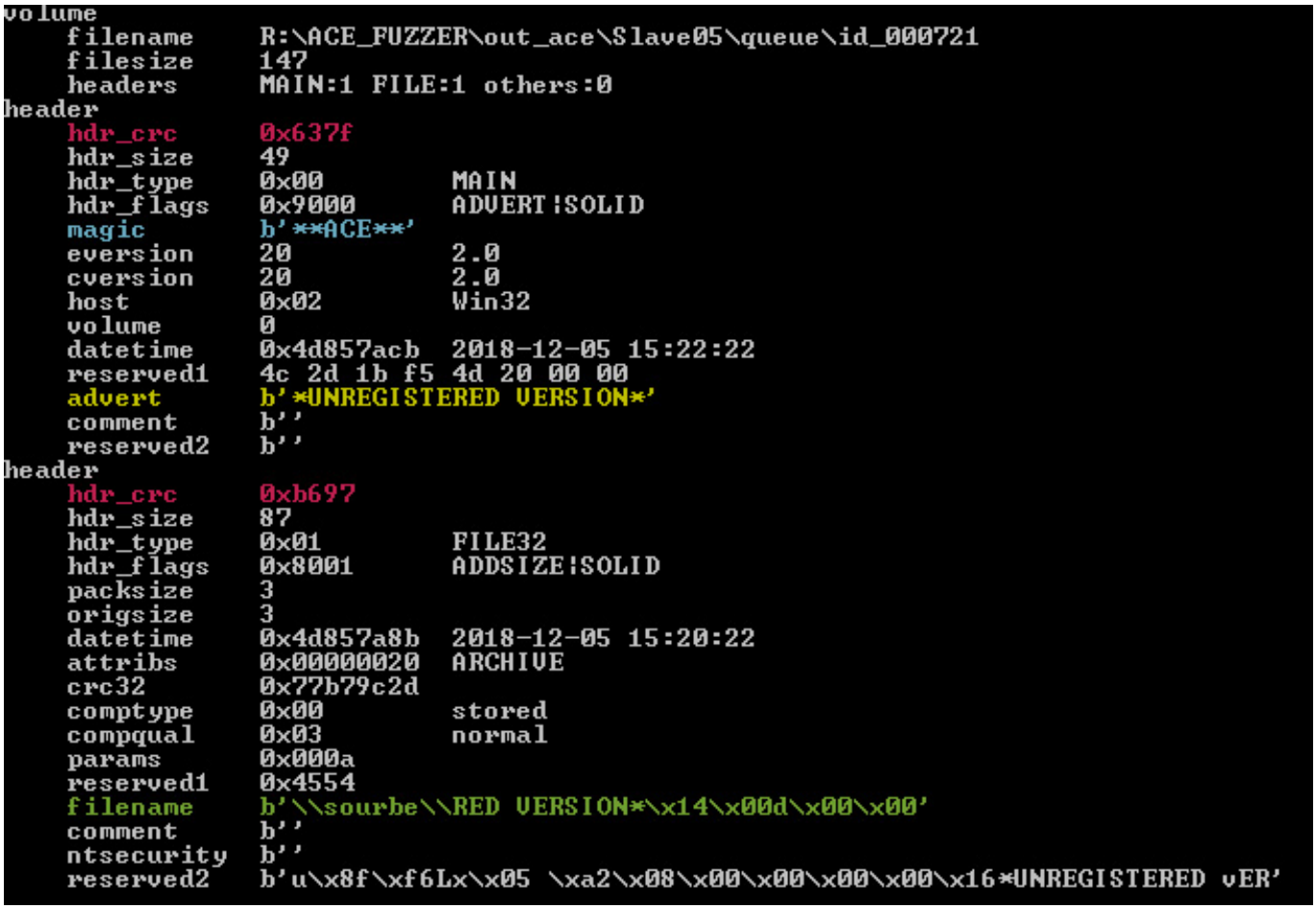
RILEVAZIONE DEL BUG PATH TRAVERSAL

Eseguendo vari tentativi con il *fuzzer*, i ricercatori hanno immediatamente rilevato un comportamento anomalo: l'output di ogni analisi dovrebbe finire all'interno della cartella "output_folders": ad esempio R:\ACE_FUZZER\\\.

È stata invece riscontrata una nuova cartella denominata *sourbe* nella root. All'interno di essa, è stato trovato un file denominato "RED VERSION_¶" con il seguente contenuto:



Analizzando l'header con acefile, si ottiene quanto segue:



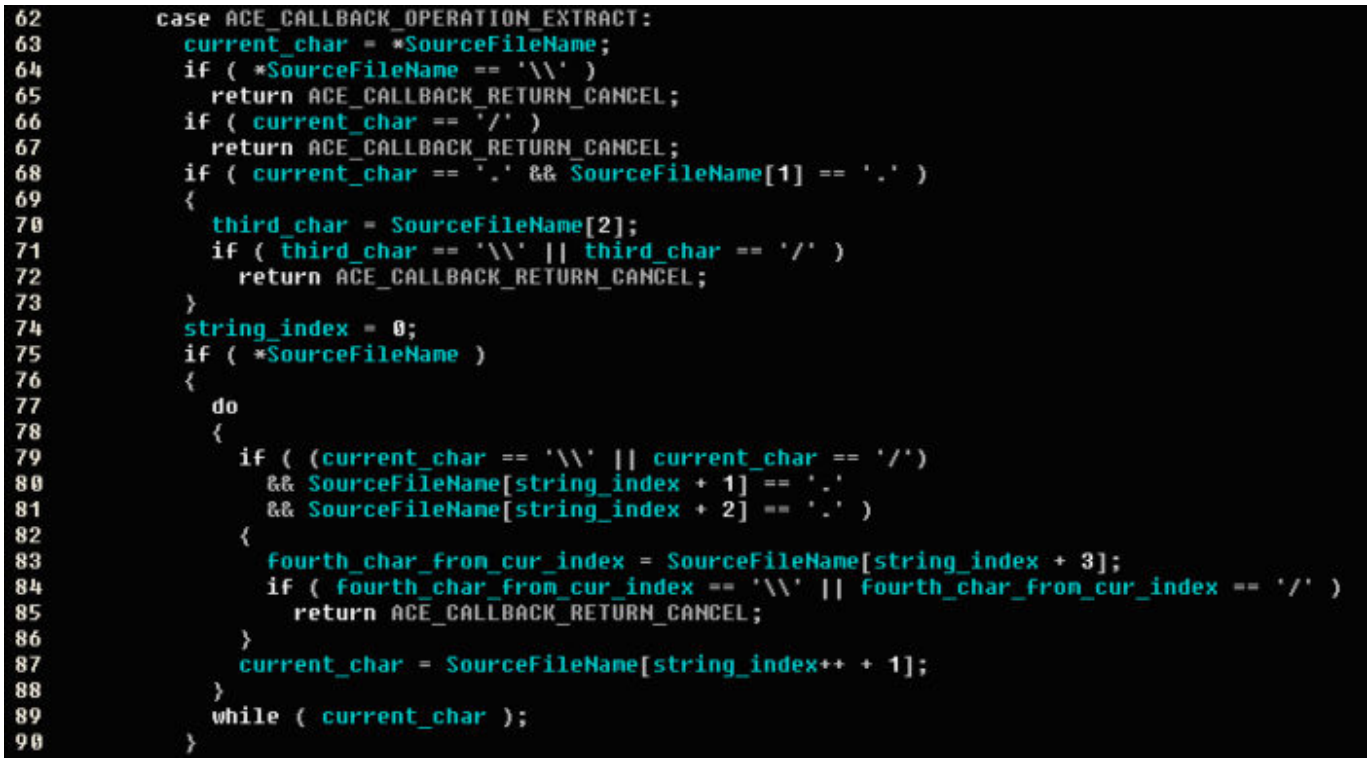
Se ne deduce che il fuzzer ha scatenato la copia di parti del campo “advert” in altri campi, nel dettaglio:

- il file è un archivio ACE;
- il contenuto dell’archivio è “SIO”, e fa parte della stringa di annuncio “*UNREGISTERED VER**SIO**N*”;
- il campo *filename* contiene la stringa “RED VERSION*” che fa parte della stringa di “*UNREGISTE RED VERSION*”;
- il percorso nel campo *filename* è stato utilizzato, nel processo di estrazione, come *path di destinazione*, sebbene faccia riferimento al *path relativo* della cartella di destinazione;
- il nome del file estratto, “RED VERSION_¶”, presuppone che:
    - gli asterischi nel campo *filename* siano convertiti nel carattere “_”;
    - il valore \ x14\ (0x14 HEX) sia il carattere di fine linea “¶”.

È quindi il momento di evidenziare quali siano stati i vincoli che hanno indotto l’algoritmo ad utilizzare il campo *filename* come *path di destinazione* per l’estrazione.

VALIDAZIONE DELL’OUTPUT

Vi sono alcune regole che l’algoritmo di estrazione applica per la validazione del *path* nel quale saranno estratti i file dell’archivio.



```
62        case ACE_CALLBACK_OPERATION_EXTRACT:
63          current_char = *SourceFileName;
64          if ( *SourceFileName == '\\' )
65            return ACE_CALLBACK_RETURN_CANCEL;
66          if ( current_char == '/' )
67            return ACE_CALLBACK_RETURN_CANCEL;
68          if ( current_char == '.' && SourceFileName[1] == '.' )
69          {
70            third_char = SourceFileName[2];
71            if ( third_char == '\\' || third_char == '/' )
72              return ACE_CALLBACK_RETURN_CANCEL;
73          }
74          string_index = 0;
75          if ( *SourceFileName )
76          {
77            do
78            {
79              if ( (current_char == '\\' || current_char == '/')
80                && SourceFileName[string_index + 1] == '.'
81                && SourceFileName[string_index + 2] == '.' )
82              {
83                fourth_char_from_cur_index = SourceFileName[string_index + 3];
84                if ( fourth_char_from_cur_index == '\\' || fourth_char_from_cur_index == '/' )
85                  return ACE_CALLBACK_RETURN_CANCEL;
86              }
87              current_char = SourceFileName[string_index++ + 1];
88            }
89            while ( current_char );
90          }
```

In figura, “SourceFileName” rappresenta il *path relativo* di estrazione.

Il validatore esegue i seguenti controlli:

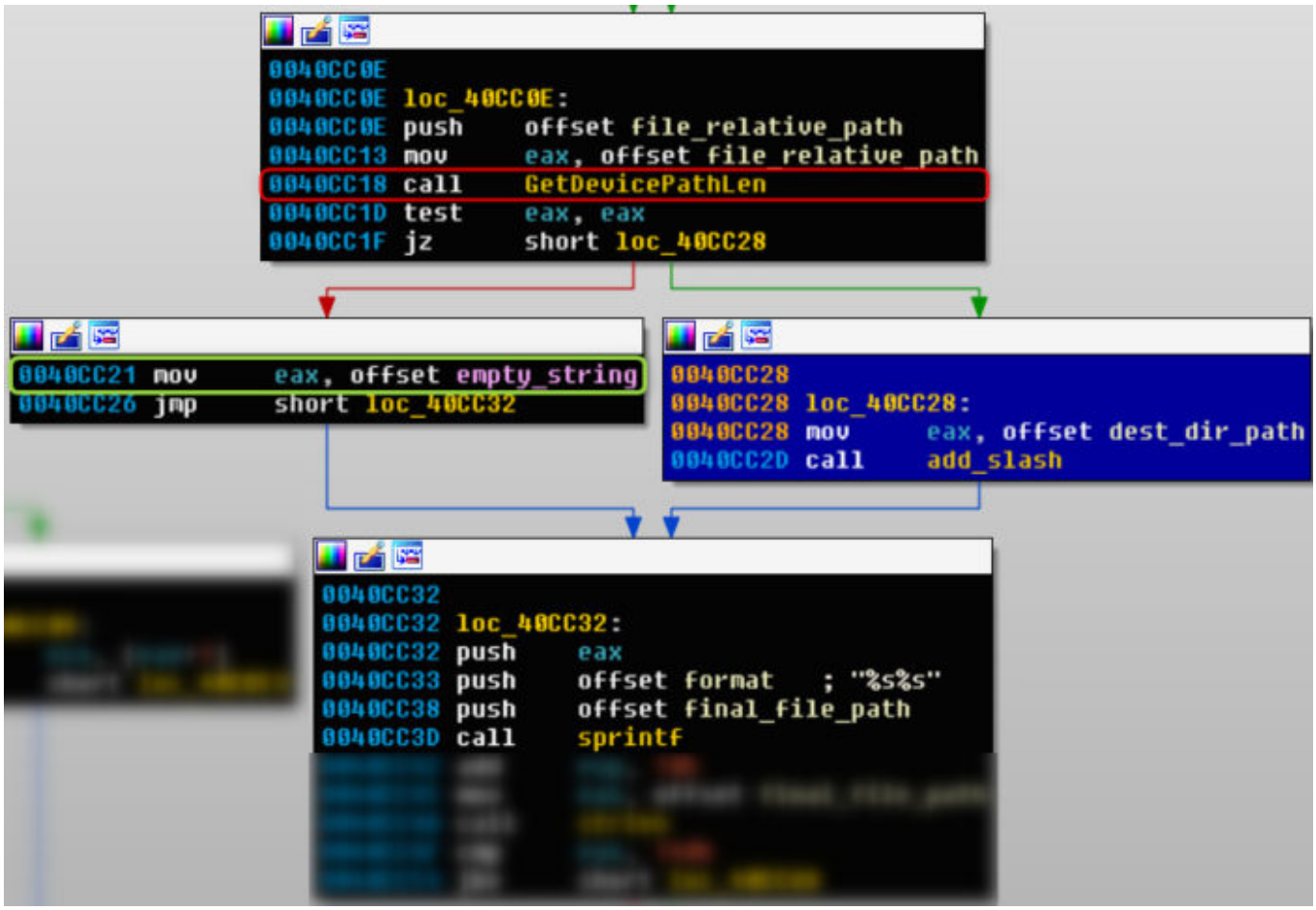
1. il primo carattere non deve essere “\” o “/”.
2. il nome del file non deve iniziare con i pattern “..\” o “../”.
3. non vi siano, all’interno della stringa, i seguenti pattern: “\..\” - “\../” - “/../” - “/..\”.

In caso contrario, l’estrazione viene interrotta.

ALLA RICERCA DEL BUG PRINCIPALE

Rimane ancora da capire perché la cartella di destinazione reale viene ignorata a favore del *path relativo* (campo *filename*).

Decompilando il codice binario, ed analizzando con IDA il flusso di esecuzione e le varie chiamate a funzione, si evince quanto segue:



L’exploit creato dal fuzzer non attraversa il blocco contrassegnato in blu, ma esegue il blocco di codice opposto (freccia rossa).

Di fatto, il codice all’interno della cornice verde sostituisce la cartella di destinazione con “” (stringa vuota); mentre la chiamata successiva alla funzione *sprintf* concatena la cartella di destinazione con il *path relativo* del file da estrarre.

Il flusso del codice è influenzato dalla chiamata alla funzione denominata *GetDevicePathLen*: se il risultato della chiamata è 0, sprintf apparirà come questo:

```
sprintf(final_file_path, "%s%s", destination_folder, file_relative_path);
```

altrimenti:

```
sprintf(final_file_path, "%s%s", "", file_relative_path);
```

È questa la porzione di codice che effettivamente attiva la vulnerabilità *Path Traversal*.

Analizziamo nel dettaglio la funzione *GetDevicePathLen* per ottenere una migliore visione del bug:

```
INT GetDevicePathLen(PCHAR Path)
{
    PCHAR      SlashPos;
    INT        Result;

    Result = 0;

    if (Path[0] == '\\')
    {
        if (Path[1] == '\\')
        {
            if (!(SlashPos = strchr(&Path[2], '\\')))
            {
                return 0;
            }

            if (!(SlashPos = strchr(SlashPos + 1, '\\')))
            {
                return 0;
            }

            Result = (UINT)SlashPos - (UINT)Path + 1;
        }
        else
        {
            Result = 1;
        }
    }
    else
    {
        if (Path[1] == ':')
        {
            Result = 2;

            if (Path[2] == '\\')
            {
                Result++;
            }
        }
    }
    return Result;
}
```

Il *path relativo* del file da estrarre viene passato a *GetDevicePathLen*.

Quest'ultima controlla che il nome del dispositivo o la lettera dell'unità siano presenti nel *path*, restituendo in output la lunghezza di tale sottostringa:

- **3** per **C:**some_folder\some_file.ext
- **1** per ****some_folder\some_file.ext

- **15** per **\\LOCALHOST\C$**some_folder\some_file.ext
- **21** per **\\?\Harddisk0Volume1**some_folder\some_file.ext
- **0** per some_folder\some_file.ext

Se il valore restituito da *GetDevicePathLen* è >0, il *path relativo* del file estratto verrà considerato come *path di destinazione*, poiché la cartella di destinazione viene sostituita da una stringa vuota durante la chiamata a sprintf.

Tuttavia, è prevista una funzione che "pulisce" il *path relativo* da eventuali tentativi di *Path Traversal*, omettendo qualsiasi sequenza non consentita prima nella chiamata *GetDevicePathLen*.

Questo è uno pseudo-codice della funzione "*CleanPath*":

```
BOOL CleanPath(PCHAR Path)
{
   char *PathTraversalPos = NULL
   if ( Path[1] == ':' && Path[2] == '\\' )
     strcpy(Path, &Path[3]);
   if ( Path[1] == ':' && Path[2] != '\\' )
     strcpy(Path, &Path[2]);
   PathTraversalPos = strstr(Path, "..\\");
   while ( PathTraversalPos )
   {
       if ( PathTraversalPos == Path || *(PathTraversalPos - 1) == '\\' )
       {
         strcpy(Path, &Path[3]);
         PathTraversalPos = strstr(Path, "..\\");
       }
       else
       {
         PathTraversalPos = strstr(Path + 1, "..\\");
       }
   }
  return Path
}
```

Tale funzione omette la sequenza "..\", se si trova all'inizio del percorso, e le occorrenze relative alle unità: "**C:**" - "**C:**" - "**C:\C:**"[5].

METTIAMO TUTTO INSIEME

L'obiettivo finale è creare un file *exploit* che faccia sì che WinRAR estragga un file su un percorso scelto arbitrariamente dall'attaccante. La destinazione ideale consiste nella cartella di *Esecuzione Automatica*, che esegue quanto al suo interno ad ogni avvio del SO.

Iniziamo col sostituire l'estensione ".ACE" in ".RAR", tanto WinRAR rileva il formato del file dal contenuto dell'header, non dall'estensione.

Sappiamo poi che ci sono almeno due cartelle di *Esecuzione Automatica* nei seguenti percorsi di Windows:

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp

C:\Users\\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

Il primo percorso richiede *privilegi elevati/livello di integrità elevato* (nel caso di UAC attivo). WinRAR viene eseguito di default con un livello di integrità medio, quindi tale soluzione non è percorribile.

Il secondo percorso richiederebbe di conoscere l'account utente.

Esiste però un vettore d'attacco migliore, che non richiede di conoscere lo .

Considerando che:

- il validatore di WinRAR e la CleanPath non ammettono i pattern:
  - \..\
  - \../
  - /../
  - /..\
  - C:\
  - C:
  - C:\C:
- la CleanPath non ammette che ad inizio stringa vi siano:
  - \
  - /
  - ..\
  - ../
- la GetDevicePathLen deve ritornare un valore >0;

utilizzando la seguente stringa nel campo *filename* nell'header dell'archivio ACE:

C:\C:C:../AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\some_file.exe

otterremo il seguente risultato finale:

C:../AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\some_file.exe

che permette di aggirare la convalida della stringa, poiché:

- la *GetDevicePathLen* restituirà 2, quindi il *path di destinazione* reale verrà scartato in favore del campo *filename*;
- *Cleanpath* rimuove "**C:\C:"**
- non vi sono pattern che rimuovono "**C:../"**, ne "**../"** è presente ad inizio stringa.

Va inoltre sottolineato che in Windows la sequenza “C:” referenzia la “directory corrente” del processo in esecuzione. Nel nostro caso la directory di default di WinRAR, tipicamente in C:\Program Files\WinRAR.

Tuttavia, interagendo con l’archivio tramite doppio clic, o facendo clic con il pulsante destro del mouse -> “Estrai”, la “directory corrente” sarà il *path* in cui risiede l’archivio, per l’appunto:

- C:\Users\\Downloads
- C:\Users\\Desktop

Quindi per arrivare alla cartella di *Esecuzione Automatica*, è bastato semplicemente salire di un livello (***../***) raggiungendo la cartella utente, senza saperne il nome, e concatenare il segmento di percorso che porta ad essa.

CONSIDERAZIONI FINALI

L’installazione di un eseguibile malevolo tramite questo exploit va a buon fine sia errori di progettazione delle librerie, sia per una serie di comportamenti imputabili all’utilizzo tipico effettuato dall’utenza.

Innanzitutto le funzioni di validazione del *path* non agiscono in maniera ricorsiva. Se così fosse, la sequenza “C:\C:C:../” sarebbe stata suddivisa in “C:\C:”, “C:” e “../”, impendendo di fatto il raggiungimento della cartella desiderata.

Inoltre, l’attaccante fa leva sulle modalità di utilizzo tipico dell’utente medio. In caso di estrazione di quest’ultimo al di fuori delle principali sottocartelle della cartella utente, tale exploit è totalmente inoffensivo.

Senza contare il fatto che l’utente predilige l’utilizzo della funzione “tasto dx -> estrai”, piuttosto che aprire l’archivio ed estrarre i file tramite GUI. In questo caso ci si renderebbe immediatamente conto che all’interno dell’archivio, oltre a file innocui, è presente un file eseguibile. E ci si renderebbe anche conto che, estratto tale archivio nelle modalità abituali, il file eseguibile non risulterebbe presente nella cartella di destinazione, evidenziando quantomeno un comportamento anomalo.

Note

[1] https://research.checkpoint.com/extracting-code-execution-from-winrar/.

[2] https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer).

[3] https://it.wikipedia.org/wiki/Directory_traversal_attack.

[4] È inoltre curioso che il creatore di tale progetto sia anche il creatore di WinRAR stesso.

[5] Si noti che non interessa la lettera dell’unità. I pattern sono validi per qualsiasi lettera.

Articolo a cura di **Daniele Rigitano**