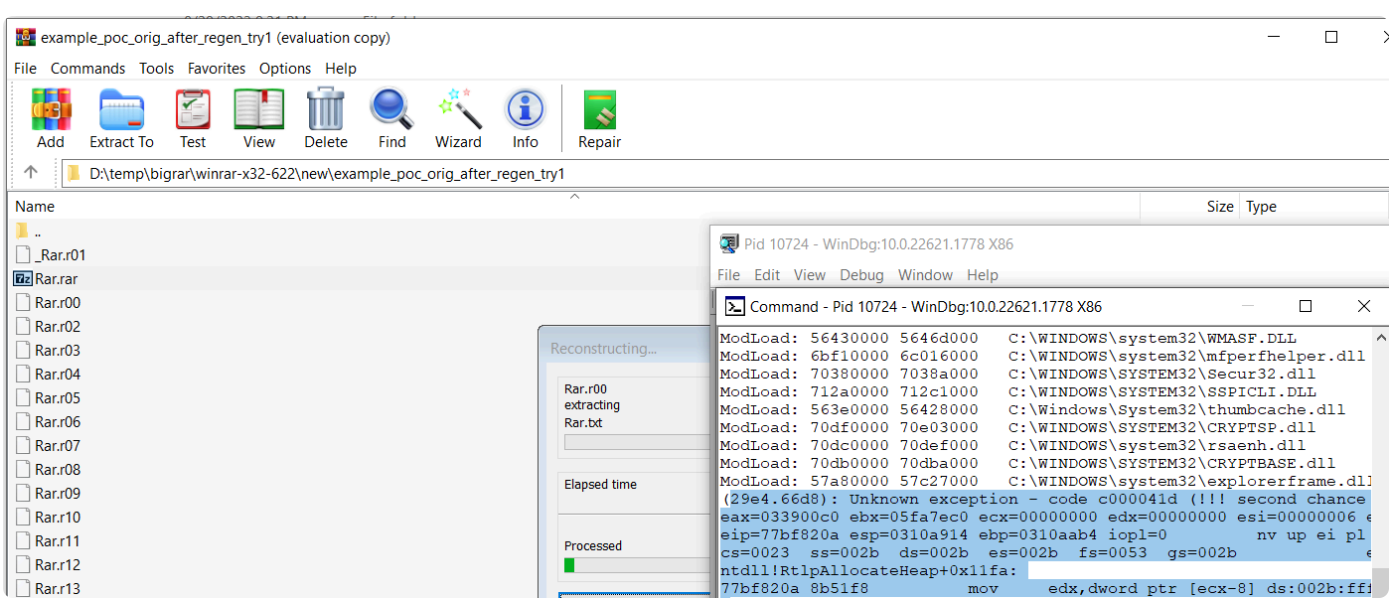
“Game of Rars” – Exploring a New Remote Code Execution Vulnerability in WinRAR with Proof-of-Concept (CVE-2023-40477) –



29
Aug

“Game of Rars” – Exploring a New Remote Code Execution Vulnerability in WinRAR with Proof-of-Concept (CVE-2023-40477) –

By Barak Sternberg | In Security Research | Comments

Executive Summary

- WinRAR, 500+ million users, is exposed to new vulnerabilities (CVE-2023-40477, CVE-2023-38831).
- Today, we present for the first-time: A PoC for CVE-2023-40477 (At time of writing)
- Although considered RCE & assumed to be exploitable, its’ impact in reality doesn’t look promising because of numerous reasons.
- We present here thorough technical research: its impact, exploitable scenarios and mitigations.
- The vulnerability was fixed in recent Winrar v6.23, we give another mitigations possibility here as well

CVE-2023-40477: Technical

It is Mid-August, Winrar 6.23 is out with some vague alerts about critical vulnerabilities.
All we know is – The vulnerability (CVE-2023-40477) is fixed and this information:

> a. Critical Bug: CVE-2023-40477. The vulnerability allows remote attackers to execute arbitrary code on affected installations. User interaction is required to exploit this vulnerability. This is fixed in the RAR4 recovery volume processing code.
>
> We would like to thank goodbyeselene in collaboration with Trend Micro Zero Day Initiative for reporting this bug.
> www.zerodayinitiative.com/advisories/ZDI-23-1152/

Armed with BinDiff, IDA & Notepad, Let's dive in.

Diffing Binaries

We know 6.23 is fixed, let's try finding minimal version before and after the fixed so we can observe just it. I've downloaded both portable winrar-v6.23 + winrar-v6.22 and observed it:

Name	Date modified	Type	Size
7zxa.dll	6/23/2023 6:07 PM	Application extens...	229 KB
Default.SFX	8/1/2023 12:26 PM	SFX File	319 KB
Default64.SFX	8/1/2023 12:26 PM	SFX File	372 KB
Descript.ion	1/4/2022 1:48 PM	ION File	2 KB
License.txt	5/22/2014 7:31 PM	Text Document	7 KB
Order.htm	8/1/2023 3:00 PM	HTML Document	4 KB
Rar.exe	8/1/2023 12:26 PM	Application	630 KB
Rar.txt	6/24/2023 1:39 PM	Text Document	110 KB
RarExt.dll	8/1/2023 12:26 PM	Application extens...	666 KB
RarExt32.dll	8/1/2023 12:26 PM	Application extens...	574 KB
RarExtInstaller.exe	8/1/2023 12:26 PM	Application	181 KB
RarExtLogo.altform-unplated_targetsize-...	10/21/2021 7:36 PM	PNG File	3 KB
RarExtLogo.altform-unplated_targetsize-...	10/21/2021 7:36 PM	PNG File	5 KB
RarExtLogo.altform-unplated_targetsize-...	10/21/2021 8:54 PM	PNG File	7 KB
RarExtPackage.msix	8/1/2023 12:25 PM	MSIX File	23 KB
RarFiles.lst	1/26/2017 7:02 PM	MASM Listing	2 KB
ReadMe.txt	7/14/2021 6:18 PM	Text Document	2 KB
Resources.pri	8/1/2023 12:25 PM	PRI File	2 KB
Uninstall.exe	8/1/2023 3:26 PM	Application	438 KB
Uninstall.lst	8/1/2023 12:26 PM	MASM Listing	1 KB
UnRAR.exe	8/1/2023 12:26 PM	Application	430 KB
WhatsNew.txt	8/1/2023 11:19 AM	Text Document	106 KB
WinCon.SFX	8/1/2023 12:26 PM	SFX File	288 KB
WinCon64.SFX	8/1/2023 12:26 PM	SFX File	342 KB
WinRAR.chm	8/1/2023 12:26 PM	Compiled HTML H...	318 KB
WinRAR.exe	8/1/2023 3:26 PM	Application	2,511 KB
Zip.SFX	8/1/2023 12:26 PM	SFX File	274 KB
Zip64.SFX	8/1/2023 12:26 PM	SFX File	313 KB

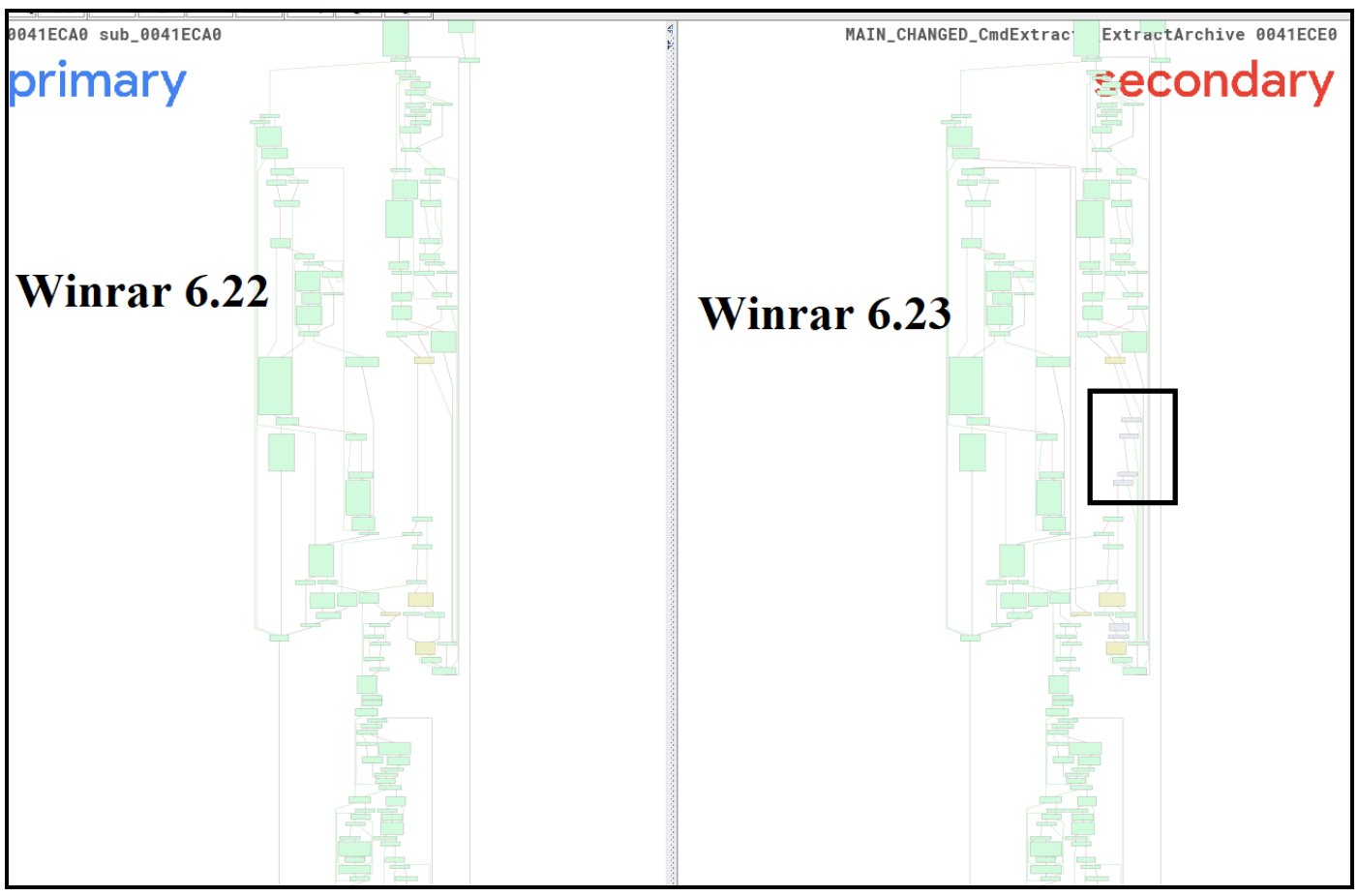
Winrar directory structure & interesting files

One can immediately observe that winrar.exe is probably just a GUI and if a vulnerability in extraction – it will probably be also in "unrar.exe". Diffing 6.23 & 6.22 gave me following diffs (ignore comments ;)):

Similarity	Confic	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.80	0.99	G------	00426200	sub_426200	00426290	sub_00426290
0.83	0.99	G------	00425FD0	sub_425FD0	00426040	sub_00426040
0.88	0.99	GI-----	004264D0	sub_4264D0	00426570	sub_00426570
0.90	0.99	G------	00448C30	loc_448C30	00448CB0	sub_00448CB0
0.90	0.99	G------	00426070	sub_426070	004260F0	sub_004260F0
0.95	0.96	GI-----	00409500	sub_409500	00409510	sub_00409510
0.97	0.99	GI-----	0041ECA0	sub_41ECA0	0041ECE0	MAIN_CHANGED_CmdExtract__ExtractArchive
0.98	0.99	GI-----	0041A250	sub_41A250	0041A260	GenArcName

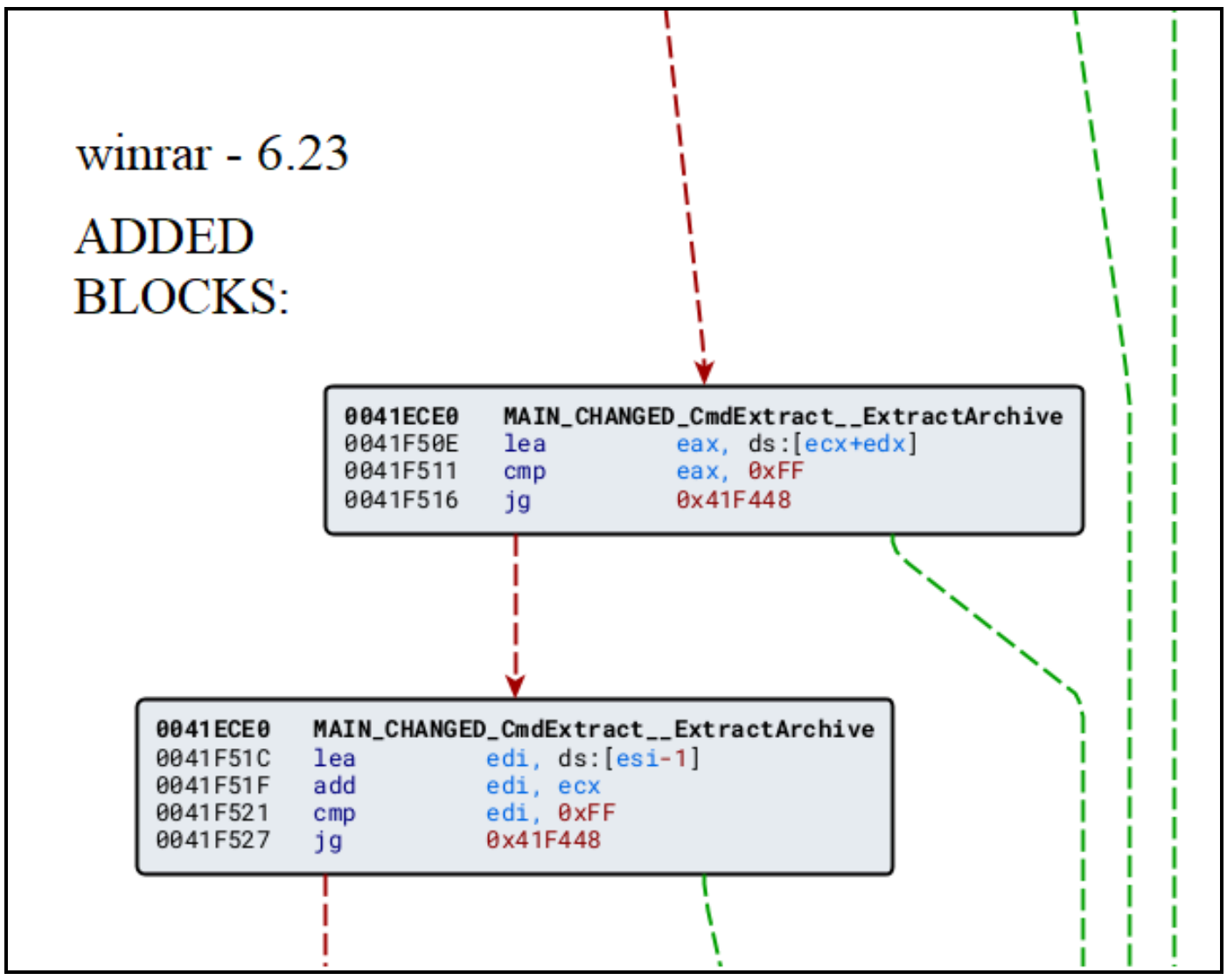
First Bindiff output

Going through flow graphs gave me couple of interesting additions in 6.23 (in big function graph – seemingly looks like the main extraction):



Highlevel bindiff additions in extract

Looking carefully seems like these are additional checks!



Bindiff's x86 interesting additional blocks

This looks promising! we see here additional checks that some var < 255 !! Interesting.
looks like overflow checks rights here.
So, time to go for a trigger (i.e change bytes to maximal values).

I have looked over RAR format, We know from the description this is related to RAR4 (vol3?) recovery volumes,
so I generated RAR4 with these features, it gave me a list of files like the following:

- RAR_FILE.rar
- RAR_FILE.r00

- RAR_FILE.r01
- RAR_FILE01.rev

I have tried to bruteforce the contents of any “byte-like” value in “.rev” & “rXX” volumes without success. it hasn’t triggered or changed anything.
Then, I tried to Google some “unrar” source-code – maybe it exists as a part of some other project?!
Success! I’ve found this old repository: https://github.com/aawc/unrar

Looking now in the source-code, We see multiple “255” const-checks, espcially in **“recvol3.cpp”**, They appear also in the original 6.22 source-code, so maybe they added extra ones because of … an overflow?
Let’s check, Diving deeper, I’ve found the security checks actually related to 0xff / 255.

The Vulnerability

unrar / recvol3.cpp

Code Blame 544 lines (483 loc) · 13.1 KB

```
215         bool WrongParam=false;
216         for (size_t I=0;I<ASIZE(P);I++)
217         {
218           do
219           {
220             Dot--;
221           } while (IsDigit(*Dot) && Dot>=CurName+BaseNamePartLength);
222           P[I]=atoiw(Dot+1);
223           if (P[I]==0 || P[I]>255)
224             WrongParam=true;
225         }
226         if (WrongParam)
227           continue;
228       }
229       if (P[1]+P[2]>255)
230         continue;
231       if (RecVolNumber!=0 && RecVolNumber!=P[1] || FileNumber!=0 && FileNumber!=P[2])
232       {
233         uiMsg(UIERROR_RECVOLDIFFSETS,CurName,PrevName);
234         return false;
235       }
236       RecVolNumber=P[1];
237       FileNumber=P[2];
238       wcsncpyz(PrevName,CurName,ASIZE(PrevName));
239       File *NewFile=new File;
240       NewFile->TOpen(CurName);
241       SrcFile[FileNumber+P[0]-1]=NewFile;
242       FoundRecVolumes++;
243
```

CVE-2023-40477 inside recvol3.cpp

Here, **P[i]’s** are extracted from the **“.rev”** files themselves. they are inside the end of files.
These **P[i]’s** are used to determine which recovery volume they represent and what **FileNumber** they suit to.
FileNumber is retrieved from them as well in **P[2]**.
Right after, **File*** is allocated and placed in an index we control in array of size 256! (line 241).
That explains the 255 checks.
Becuase the index is actually equals to:**P[2]+P[0]-1**. We can almost arbitrarily control it with “rev” volumes contents.
So, after all, we get to overwrite that buffer with pointers (pointing to **File** structures), they are overwriting the next properties in the current object.

To trigger that vulnerability, we found “Restore()” needed to be called in recvol3.cpp.
We found out we also need the reconstruction step to happen, so something will be done with the overwrite of pointers.

For that to happen, one needs to have some missing volumes of rar (.r00 missing), and .rev volumes available.
Also, we need to make sure crc32 checksums are correct, this concluded to just couple of lines to trigger.
For our ease, We have used the original rar4 recovery volumes we generated with the GUI, but this can surely be much smaller and a more efficient trigger with probably up to 2-3 files at most.

```
# 1. re-generate malformed recovery vols.
data = open('%s01.rev' % ARCHIVE_NAME, 'rb').read() # just use the first and
malform it up.
names = ['%s%s.rev' % (ARCHIVE_NAME, str(i).zfill(2)) for i in range(256)]
# "destroy" the P[i]'s
datas = [data[:-7] + bytes([0xf0, 0x00, i]) + calc_crc(data[:-7] + bytes([0xf0,
0x00, i])) for i in range(256)]

# 2. overwrite malformed recovery vols.
for i in range(256):
        fname = names[i]
        data = datas[i]
        open(fname, 'wb').write(data)
```

Our PoC @ https://github.com/wildptr-io/Winrar-CVE-2023-40477-POC/

Shortly after finding it, We’ve also seen https://www.rarlab.com/vuln_rev3_names.html in rar-labs site, confirming in high probability our findings.

Using that, we have successfully crashed winrar / unrar in couple of scenarios:

1. Memset overwrite with zeroes to invalid memory (probably after the buffer):

Crash while using winrar’s unrar.exe – on memset of *Buf*

from source-code – the memset of *Buf*

2. Heap overflow possible scenario in winrar.exe when using “extract to”

Heap Overflow in winrar.exe – possible scenario

Exploitable?

To determine exploitability, let’s look at structures we override after the 256-array.
Let’s determine how attackers can use this primitive to gain RCE & what mitigations

```
// RecVolume3 struct - that gets overflowed
class RecVolumes3
{
  private:
    File *SrcFile[256]; // overflow in here with File* pointers.
    Array Buf;

#ifdef RAR_SMP
    ThreadPool *RSThreadPool;
#endif
  public:
    RecVolumes3(CommandData *Cmd,bool TestOnly);
    ~RecVolumes3();
    void Make(CommandData *Cmd,wchar *ArcName);
    bool Restore(CommandData *Cmd,const wchar *Name,bool Silent);
    void Test(CommandData *Cmd,const wchar *Name);
};
...
...
// Array template class:
template  class Array
{
  private:
    T *Buffer;
    size_t BufSize;
    size_t AllocSize;
    size_t MaxSize;
  public:
    Array();
    Array(size_t Size);
    Array(const Array &Src); // Copy constructor.
    ~Array();
```

So, after all, **We overflow “Buffer” object with “File*” pointers.**
That’s nice, but not enough, one would probably need to bypass many defenses enabled here.
Fortunately, Many protections are inside: ASLR, CFG, Stack-Cookie & DEP.
That is also without mentioning the binary differences between winrar versions. All of that makes exploitation possible but not so probable for the average threat-actor.

Impact

Despite its high-severity CVE rating, the complexity required for successful exploitation suggests a low likelihood of widespread abuse, unlike, for example, the widely exploited Log4j RCE vulnerability.

Interestingly, Chromium appears to utilize the unrar library as well:https://chromium.googlesource.com/chromium/src/+/6ff23b0604e2edbe7ef282564ea340f5c72ab91a%5E%21/. It’s plausible that the unrar library is incorporated into Chrome OS as a third-party dependency, although code references suggesting its usage are absent.

Mitigations

In terms of mitigation strategies, there are several feasible options to consider:

- Full fix: Update Winrar to 6.23+ will fully fix that.
- Patch-Up: If update is not available or hard to deploy – **blocking *.rev files (best to block also: *.rXX files and *.partXX.rar).** This might be a quick fix-up although not fully asserted nor official.

Additional Resources & References:

CVE-2023-40477 in the news – The Register,

CVE-2023-40477 report,

RAR-Labs extra explanation for CVE-2023-40477 in 6.23 fix

Tags: CVE-2023-40477, Mitigations, Security Research, Vulnerability, Winrar CVE
