Extracting a 19 Year Old Code Execution from WinRAR

February 20, 2019

Research by: Nadav Grossman

Introduction

In this article, we tell the story of how we found a logical bug using the WinAFL fuzzer (https://github.com/googleprojectzero/winafl) and exploited it in WinRAR (https://www.win-rar.com/start.html?&L=0) to gain full control over a victim's computer. The exploit works by just extracting an archive, and puts over 500 million users (https://www.win-rar.com/start.html?&L=0) at risk. This vulnerability has existed for over 19 years(!) and forced WinRAR to completely drop support for the vulnerable format.

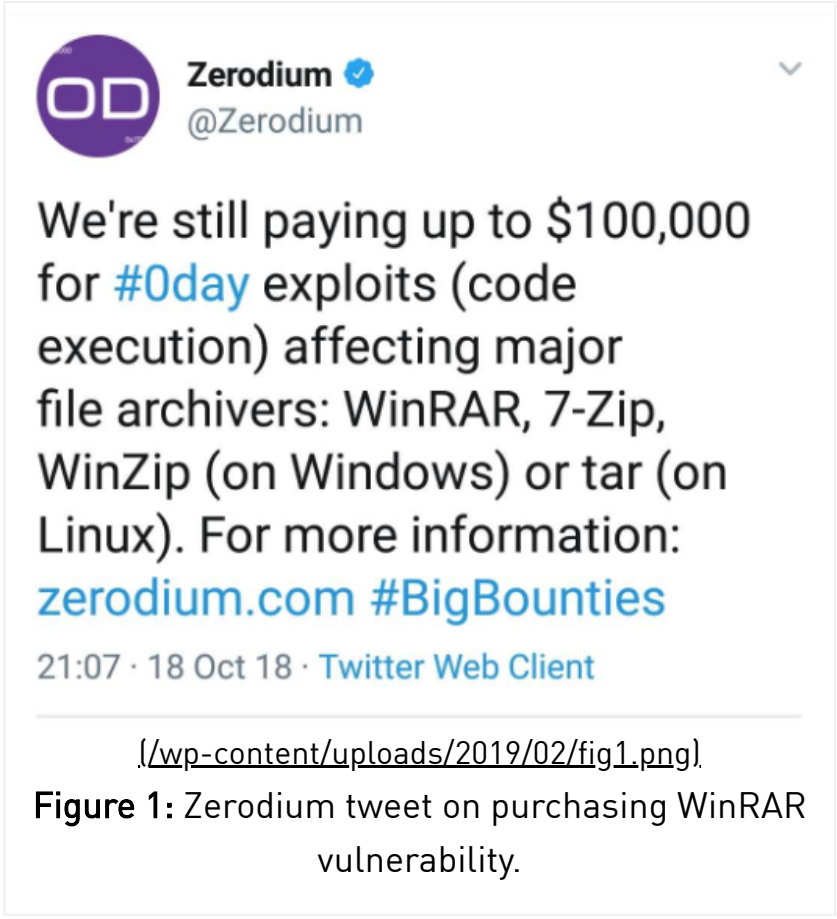
Background

A few months ago, our team built a multi-processor fuzzing lab and started to fuzz binaries for Windows environments using the WinAFL fuzzer. After the good results we got from our Adobe Research (https://research.checkpoint.com/50-adobe-cves-in-50-days/), we decided to expand our fuzzing efforts and started to fuzz WinRAR too.

One of the crashes produced by the fuzzer led us to an old, dated dynamic link library (dll) that was compiled back in 2006 without a protection mechanism (like ASLR, DEP, etc.) and is used by WinRAR.

We turned our focus and fuzzer to this "low hanging fruit" dll, and looked for a memory corruption bug that would hopefully lead to Remote Code Execution.
However, the fuzzer produced a test case with "weird" behavior. After researching this behavior, we found a logical bug: Absolute Path Traversal. From this point on it was simple to leverage this vulnerability to a remote code execution.

Perhaps it's also worth mentioning that a substantial amount of money in various bug bounty programs is offered for these types of vulnerabilities.



(/wp-content/uploads/2019/02/fig1.png)

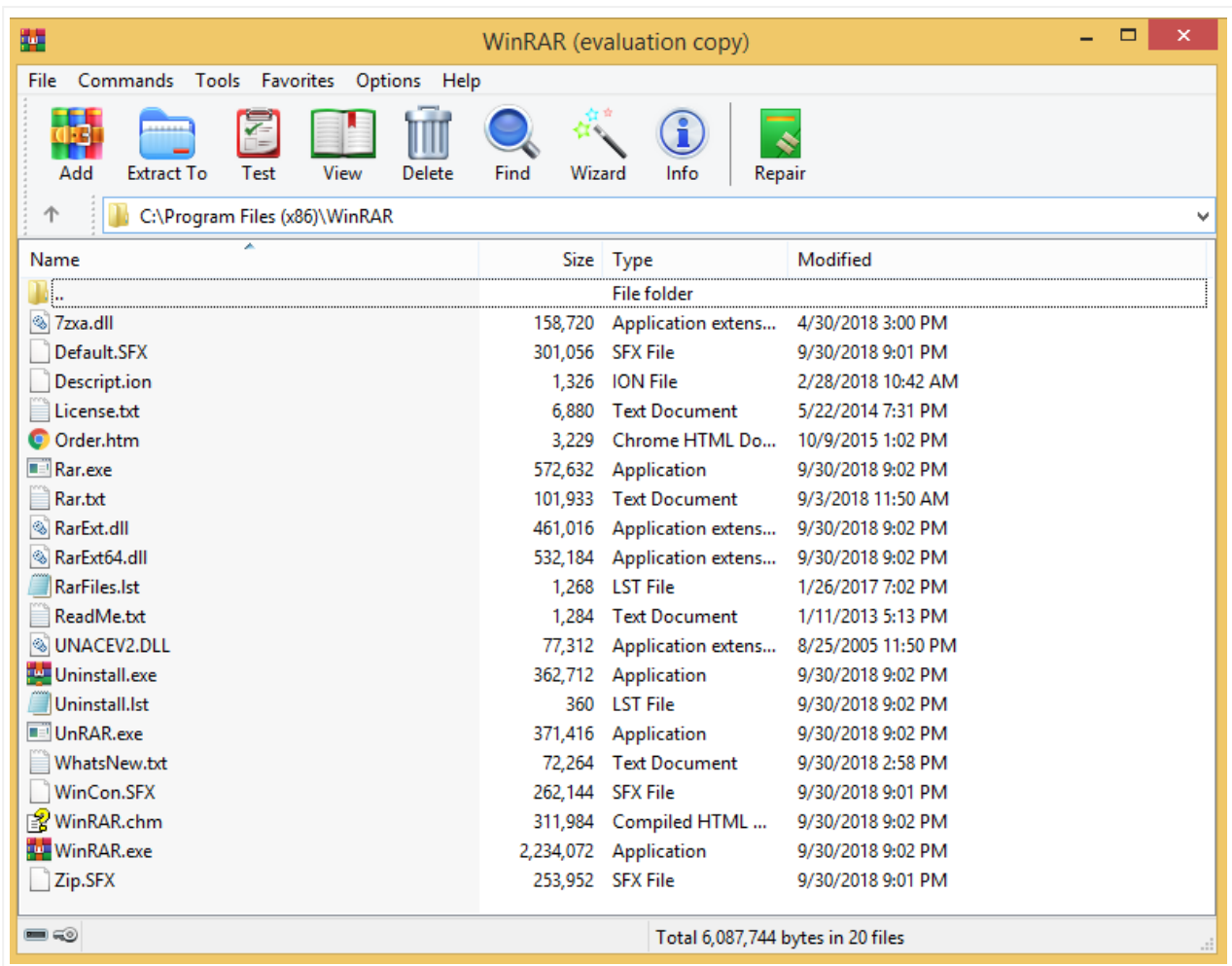
Figure 1: Zerodium tweet on purchasing WinRAR vulnerability.

What is WinRAR?

WinRAR is a trialware file archiver utility for Windows which can create and view archives in RAR or ZIP file formats and unpack numerous archive file formats.

According to the WinRAR website (https://www.win-rar.com/start.html?&L=0), over 500 million users worldwide make WinRAR the world's most popular compression tool today.

This is what the GUI looks like:



(/wp-content/uploads/2019/02/fig2.png)

Figure 2: WinRAR GUI.

The Fuzzing Process Background

These are the steps taken to start fuzzing WinRAR:

1. Creation of an internal harness inside the WinRAR main function which enables us to fuzz any archive type, without stitching a specific harness for each format. This is done by patching the WinRAR executable.
2. Eliminate GUI elements such as message boxes and dialogs which require user interaction. This is also done by patching the WinRAR executable.
   There are some message boxes that pop up even in CLI mode of WinRAR.
3. Use a giant corpus from an interesting piece of research conducted around 2005 by the University of Oulu (https://www.ee.oulu.fi/roles/ouspg/PROTOS_Test-Suite_c10-archive).
4. Fuzz the program with WinAFL using WinRAR command line switches. These force WinRAR to parse the "broken archive" and also set default passwords ("-p" for password and "-kb" for keep broken extracted files). We found those options in a WinRAR manual/help file.

After a short time of fuzzing, we found several crashes in the extraction of several archive formats such as RAR, LZH and ACE that were caused by a memory corruption vulnerability such as Out-of-Bounds Write. The exploitation of these vulnerabilities, though, is not trivial because the primitives supplied limited control over the overwritten buffer.

However, a crash related to the parsing of the ACE format caught our eye. We found that WinRAR uses a dll named `unacev2.dll` for parsing ACE archives. A quick look at this dll revealed that it's an old dated dll compiled in 2006 without a protection mechanism. In the end, it turned out that we didn't even need to bypass them.

Build a Specific Harness

We decided to focus on this dll because it looked like it would be quick and easy to exploit.

Also, as far as WinRAR is concerned, as long as the archive file has a *.rar* extension, it would handle it according to the file's magic bytes, in our case – the ACE format.

To improve the fuzzer performance, and to increase the coverage only on the relevant dll, we created a specific harness for `unacev2.dll` .

To do that, we need to understand how `unacev2.dll` is used. After reverse engineering the code calling `unacev2.dll` for ACE archive extraction, we found that two exported functions should be called for extraction in the following order:

1. An initialization function named `ACEInitDll`, with the following signature:
   ```
   INT __stdcall ACEInitDll(unknown_struct_1 *struct_1);
   ```
   • struct_1: pointer to an unknown struct
2. An extraction function named `ACEExtract` , with the following signature:
   ```
   INT __stdcall ACEExtract(LPSTR ArchiveName, unknown_struct_2 *struct_2);
   ```
   •`ArchiveName`: string pointer to the path to the ace file to be extracted
   •`struct_2`: pointer to an unknown struct

Both of these functions required structs that are unknown to us. We had two options to try to understand the unknown struct: reversing and debugging WinRAR, or trying to find an open source project that uses those structs.

The first option is more time consuming, so we opted to try the second one. We searched github.com for the exported function `ACEInitDll`
and found a project named FarManager (https://github.com/FarGroup/FarManager) that uses this dll and includes a detailed header file for the unknown structs.
Note: The creator of this project is also the creator of WinRAR.

After loading the header files to IDA, it was much easier to understand the previously "unknown structs" to both functions (`ACEInitDll` and `ACEExtract`), as IDA displayed the correct name and type for each struct member.

From the headers we found in the FarManager project, we came up with the following signature:

```
INT __stdcall ACEInitDll(pACEInitDllStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/UNACEFNC.H#LC90) DllData);
```

```
INT __stdcall ACEExtract(LPSTR ArchiveName, pACEExtractStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/UNACEFNC.H#LC194) Extract);
```

To mimic the way that WinRAR uses `unacev2.dll` , we assigned the same struct member just as WinRAR did.

We started to fuzz this specific harness, but we didn't find new crashes and the coverage did not expand in the first few hours of the fuzzing. We tried to understand the reason for this limitation.

We started by looking for information about the ACE archive format.

Understanding the ACE Format

We didn't find a RFC for that format, but we did find vital information over the internet.

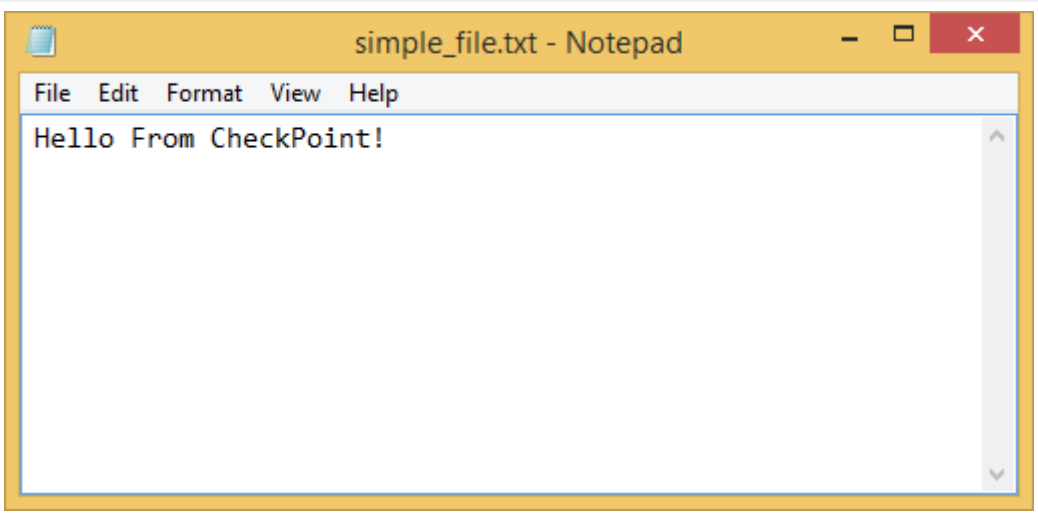
1. Creating an ACE archive is protected by a patent. The only software that is allowed to create an ACE archive is `WinACE` (https://web.archive.org/web/20170714193504/http:/winace.com:80/). The last version of this program was compiled in November 2007. The company's website has been down since August 2017. However, extracting an ACE archive is not protected by a patent.

2. A pure Python project named `acefile` (https://pypi.org/project/<code>acefile</code> /) is mentioned in this Wikipedia page (https://en.wikipedia.org/wiki/ACE_(compressed_file_format)). Its most useful features are:

- It can extract an ACE archive.
- It contains a brief explanation about the ACE file format.
- It has a very helpful feature that prints the file format header with an explanation.

To understand the ACE file format, let's create a simple `.txt` file (named "simple_file.txt"), and compress it using `WinACE`. We will then check the headers of the ACE file using `acefile` .

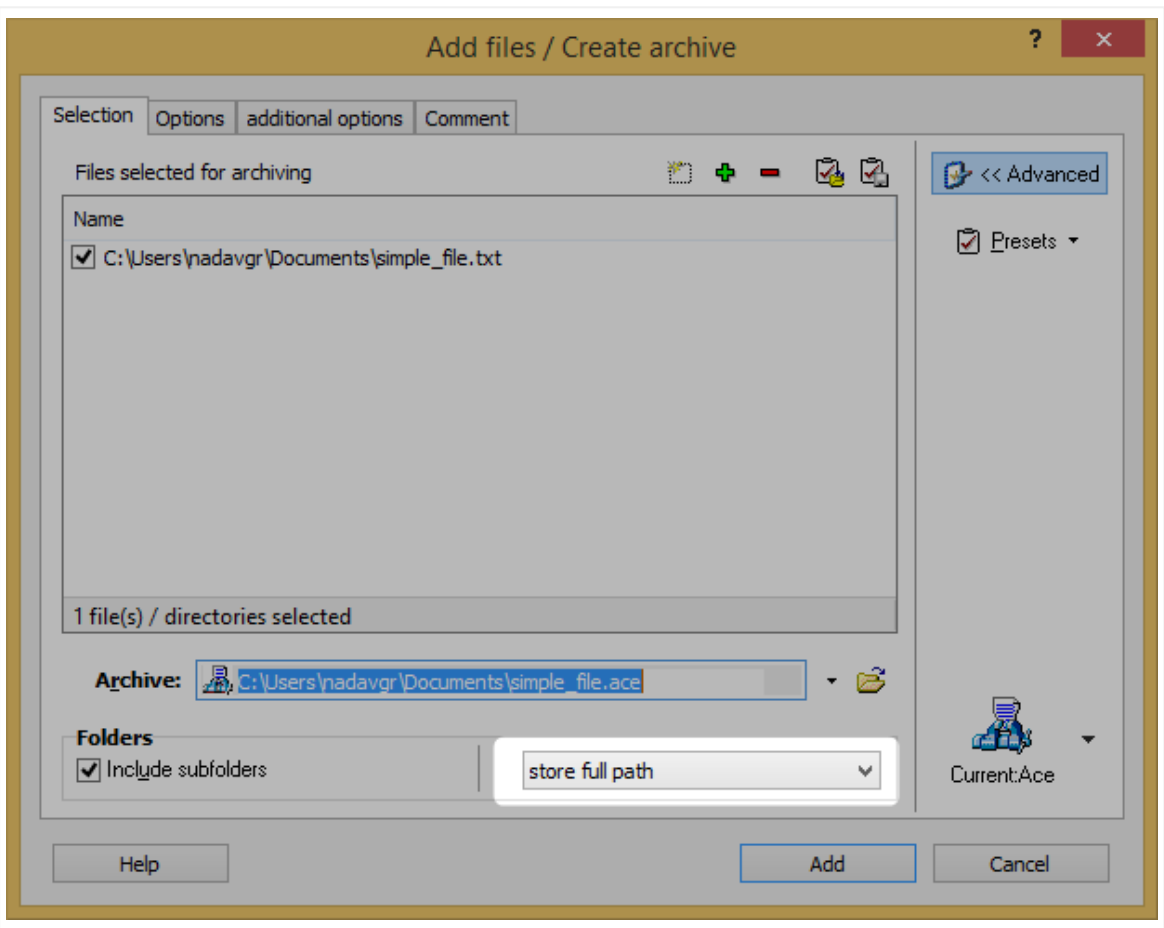
This is **`simple_file.txt`**



(/wp-content/uploads/2019/02/fig3.png)

Figure 3: File before compression.

These are the options we selected in `WinACE`to create our example:

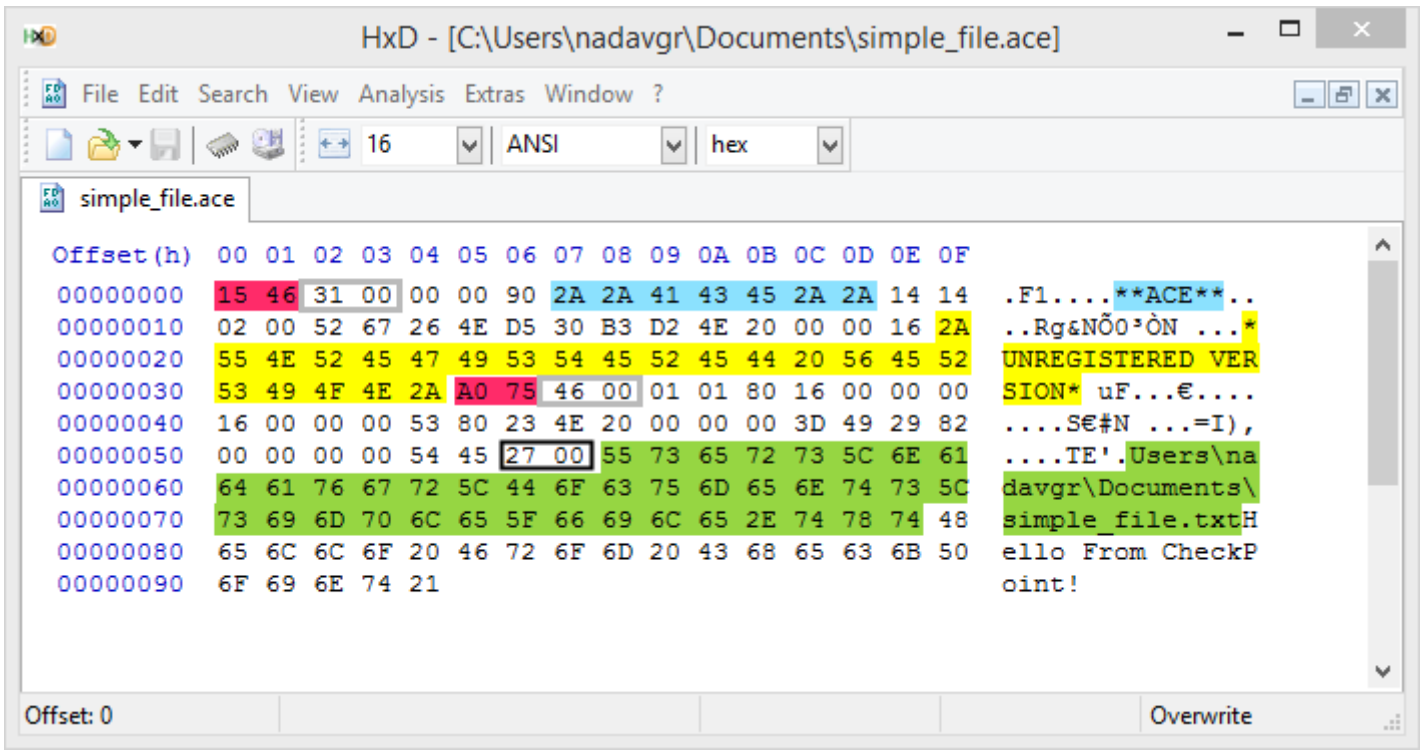


(/wp-content/uploads/2019/02/fig4.png)

Figure 4: WinACE compression GUI.

This option creates the subdirectories `\users\nadavgr\Documents` under the chosen extraction directory and extracts simple_file.txt to that relative path.

simple_file.ace



(/wp-content/uploads/2019/02/fig5.png)

Figure 5: The simple_file.ace produced using WinACE's "store" compression option for visibility.

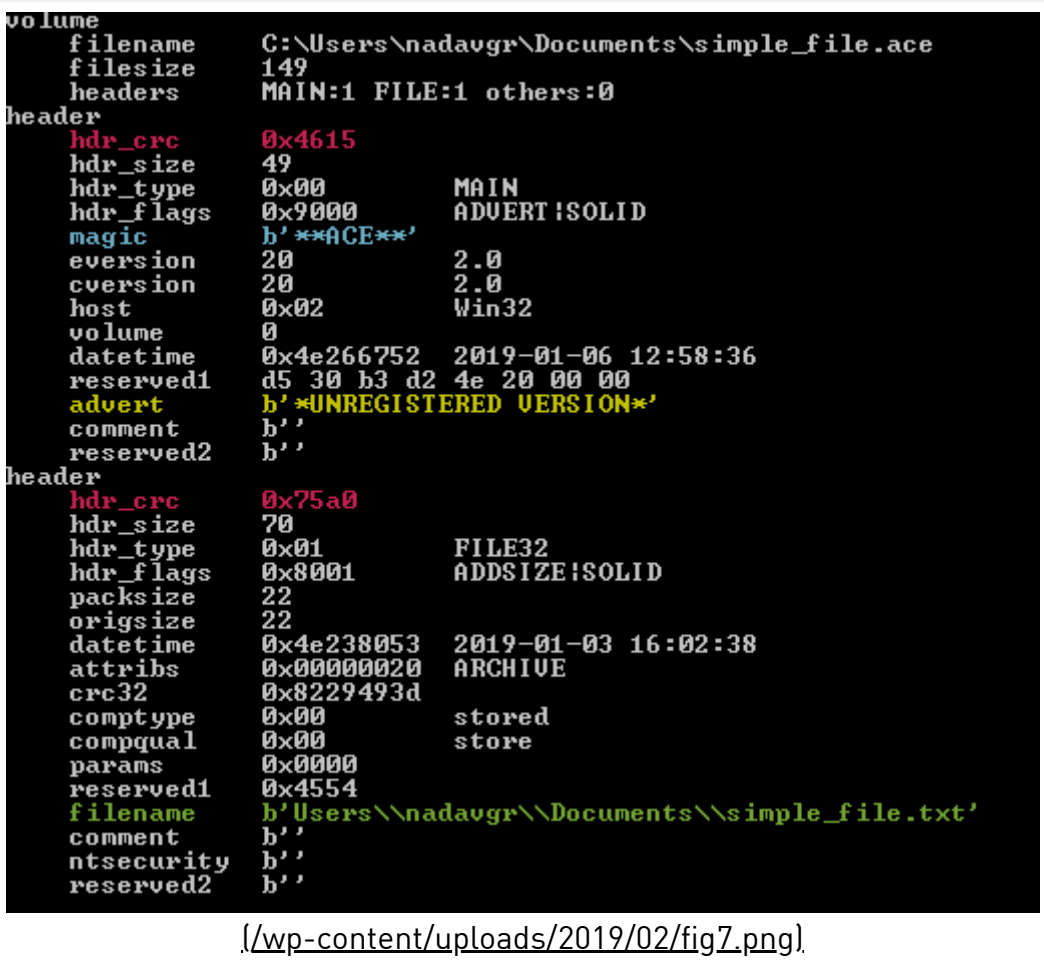
Running `acefile.py` from the `acefile` (https://pypi.org/project/<code>acefile</code> /) project using headers flags displays information about the archive headers:

```
py acefile.py --headers "C:\Users\nadavgr\Documents\simple_file.ace"
```

(/wp-content/uploads/2019/02/fig6.png)

Figure 6: Parsing ACE file header using acefile.py.

This results in:



(/wp-content/uploads/2019/02/fig7.png)

Figure 7: acefile.py header parsing output.

Notes:

- Consider each "\\" from the `filename` field in the image above as a single slash "\", this is just python escaping.
- For clarity, the same fields are marked with the same color in the hex dump and in the output from`acefile`.

Summary of the important fields:

- **hdr_crc (marked in pink):**
  Two CRC fields are present in 2 headers. If the CRC doesn't match the data, the extraction

is interrupted. This is the reason why the fuzzer didn't find more paths (expand its coverage). To "solve" this issue we patched all the CRC* checks in `unacev2.dll` .*Note – The CRC is a modified implementation (https://github.com/droe/acefile/blob/master/acefile.py#LC868) of the regular CRC-32 (https://en.wikipedia.org/wiki/Cyclic_redundancy_check#CRC-32_algorithm).

- **filename (marked in green):**
  It contains the relative path to the file. All the directories specified in the relative path are created during the extracting process (including the file). The size of the `filename` is defined by 2 bytes (little endian) marked by a black frame in the hex dump.
- **advert (marked in yellow)**
  The advert field is automatically added by `WinACE`, during the creation of an ACE archive, if the archive is created using an unregistered version of `WinACE`.
- **file content:**
  - "`origsize` " – The content's size. The content itself is positioned after the header that defines the file ("hdr_type" field == 1).
  - "`hdr_size` " – The header size. Marked by a gray frame in the hex dump.
  - At offset 70 (0x46) from the second header, we can find our file content: "Hello From Check Point!"

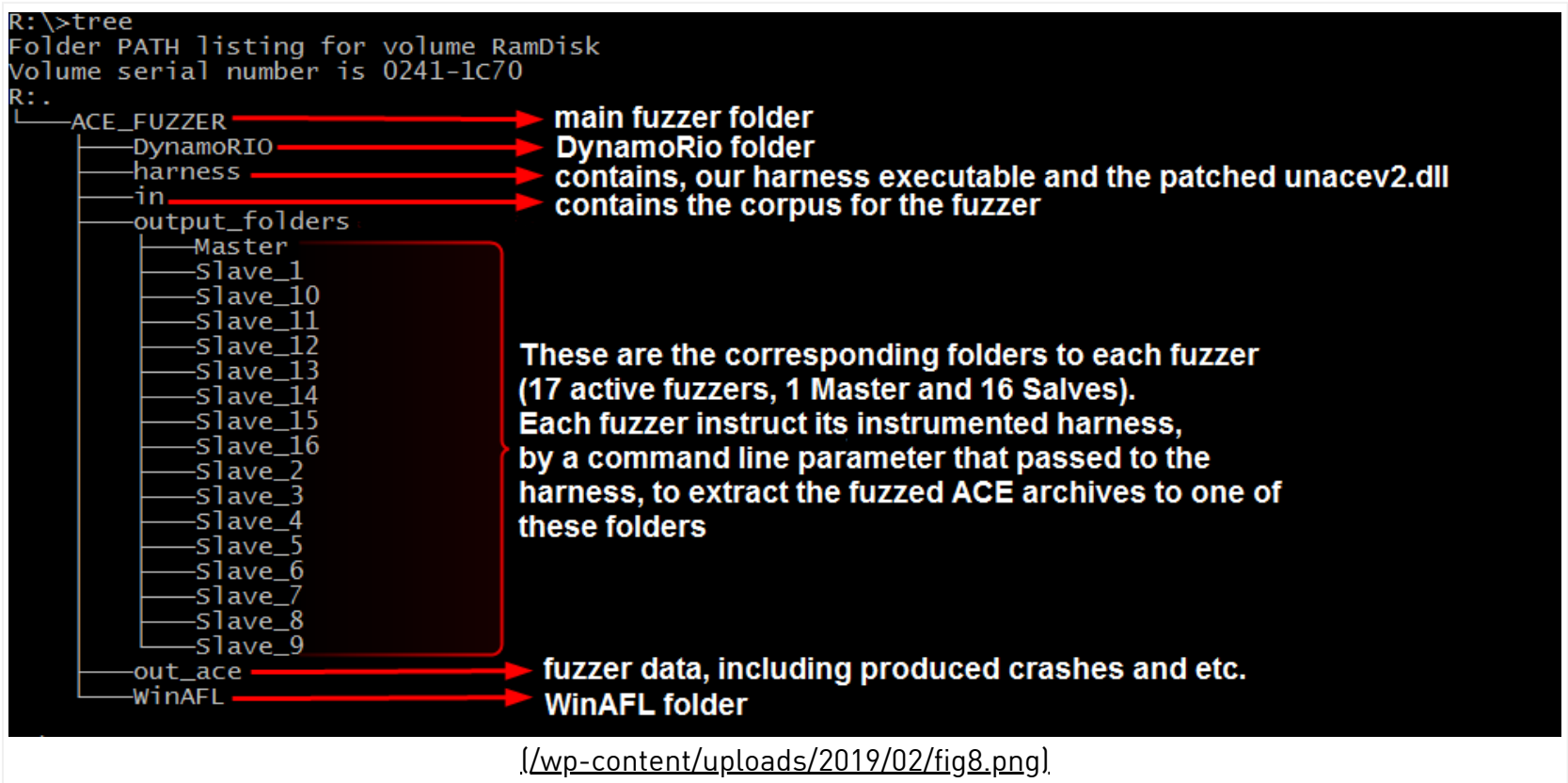
Because the filename field contains the relative path to the file, we did some manual modification attempts to the field to see if it is vulnerable to "Path Traversal."
For example, we added the trivial path traversal gadget "\..\" to the filename field and more complex "Path Traversal" tricks as well, but without success.

After patching all the structure checks, such as the CRC validation, we once again activated our fuzzer. After a short time of fuzzing, we entered the main fuzzing directory and found something odd. But let's first describe our fuzzing machine for some necessary background.

The Fuzzing Machine

To increase the fuzzer performance and to prevent an I\O bottleneck, we used a RAM disk drive that uses the ImDisk toolkit (https://sourceforge.net/projects/imdisk-toolkit/) on the fuzzing machine.

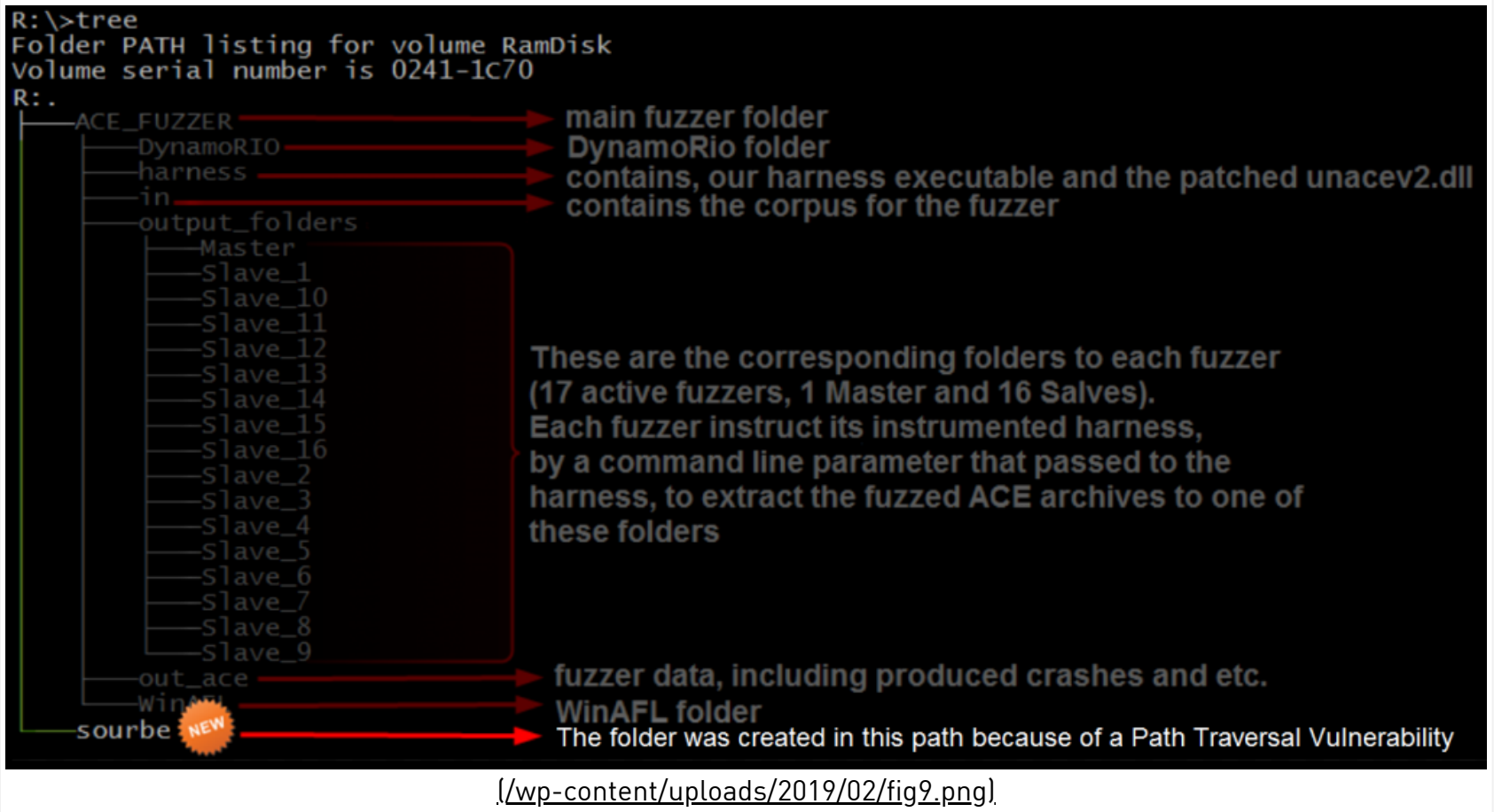
The Ram disk is mapped to drive **R:**, and the folder tree looks like this:



(/wp-content/uploads/2019/02/fig8.png)
Figure 8: Fuzzer's folders hierarchy

Detecting the Path Traversal Bug

A short time after starting the fuzzer, we found a new folder named sourbe in a surprising location, in the root of drive **R:**

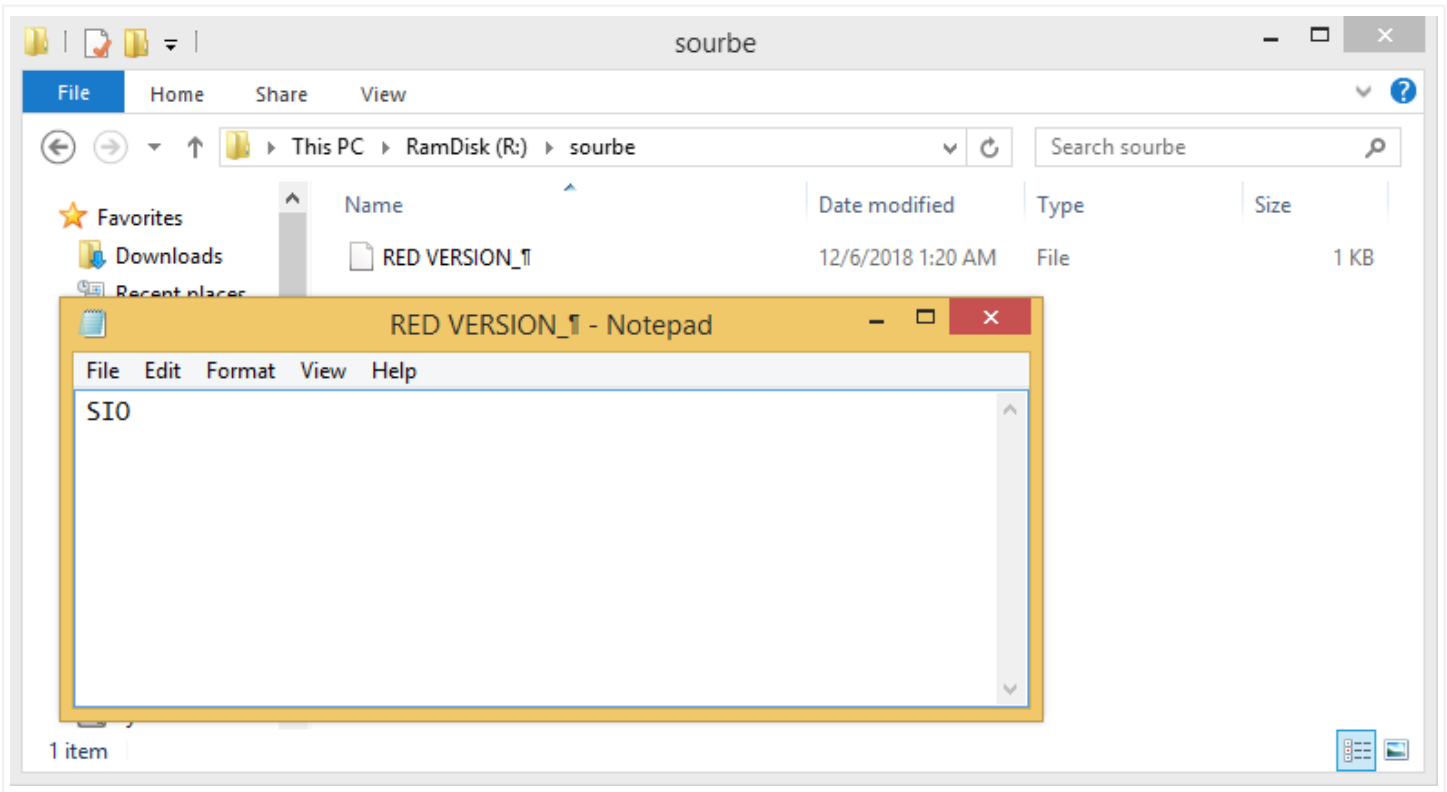


(/wp-content/uploads/2019/02/fig9.png)

Figure 9: "sourbe", the unexpected folder which created during fuzzing.

The harness is instructed to extract the fuzzed archive to sub-directories under "output_folders". For example, `R:\ACE_FUZZER\output_folders\Slave_2\` . So why do we have a new folder created in the parent directory?

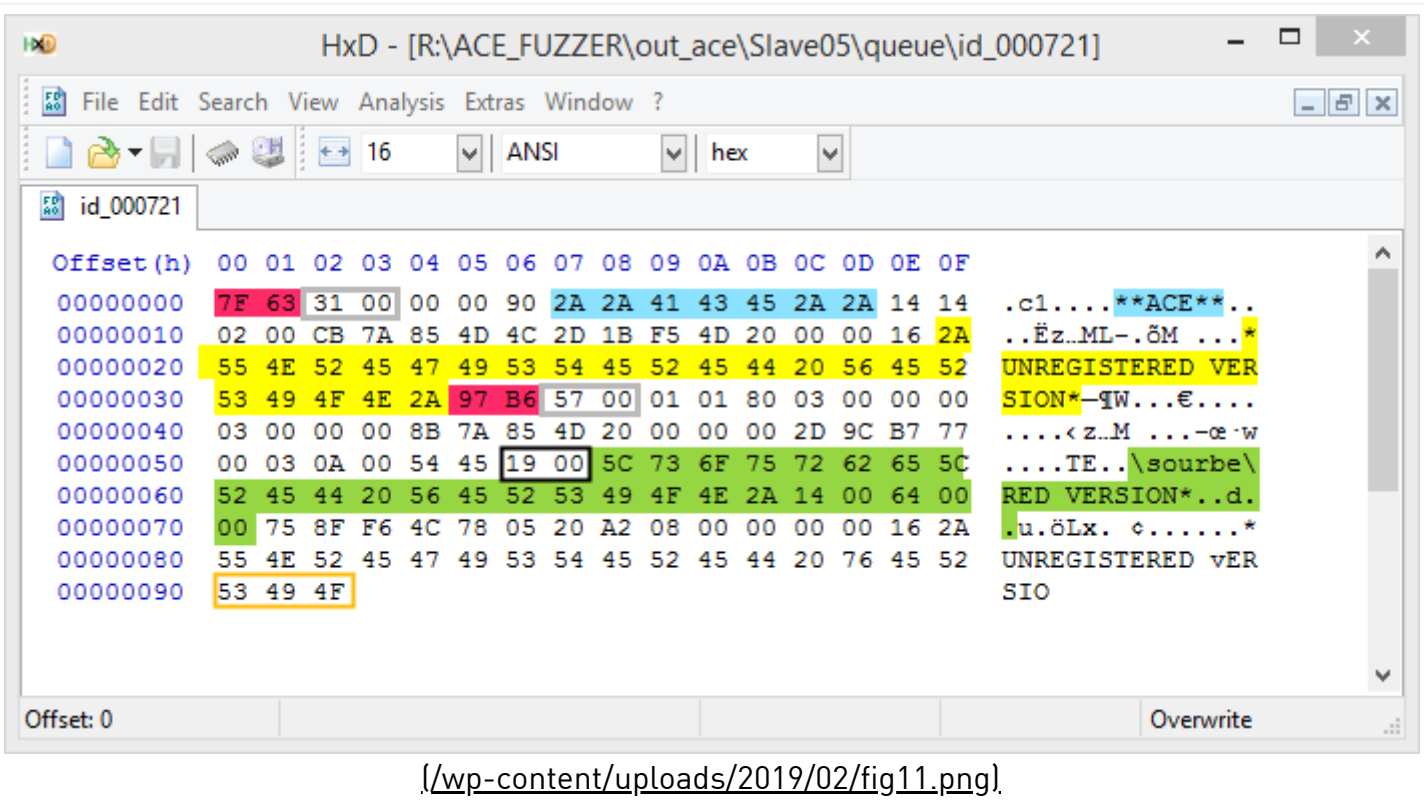
Inside the `sourbe` folder we found a file named `RED VERSION_¶` with the following content:



(/wp-content/uploads/2019/02/fig10.png)

Figure 10: Content of the file that produced by the fuzzer in the unexpected path "R:\sourbe\RED VERSION_¶".

This is the hex dump of the test case that triggers the vulnerability:



(/wp-content/uploads/2019/02/fig11.png)

Figure 11: A hex dump of the file that produced by the fuzzer in the unexpected path "R:\sourbe\RED VERSION_¶".

Notes:

- We made some minor changes to this test case, (such as adjusting the CRC) to make it parsable by `acefile`.
- For convenience, fields are marked with the same color in the hex dump and in the output from `acefile`.

```
volume
    filename    R:\ACE_FUZZER\out_ace\Slave05\queue\id_000721
    filesize    147
    headers     MAIN:1 FILE:1 others:0
header
    hdr_crc     0x637f
    hdr_size    49
    hdr_type    0x00        MAIN
    hdr_flags   0x9000      ADVERT!SOLID
    magic       b'**ACE**'
    eversion    20          2.0
    cversion    20          2.0
    host        0x02        Win32
    volume      0
    datetime    0x4d857acb  2018-12-05 15:22:22
    reserved1   4c 2d 1b f5 4d 20 00 00
    advert      b'*UNREGISTERED VERSION*'
    comment     b''
    reserved2   b''
header
    hdr_crc     0xb697
    hdr_size    87
    hdr_type    0x01        FILE32
    hdr_flags   0x8001      ADDSIZE!SOLID
    packsize    3
    origsize    3
    datetime    0x4d857a8b  2018-12-05 15:20:22
    attribs     0x00000020  ARCHIVE
    crc32       0x77b79c2d
    comptype    0x00        stored
    compqual    0x03        normal
    params      0x000a
    reserved1   0x4554
    filename    b'\\sourbe\\RED VERSION*\x14\x00d\x00\x00'
    comment     b''
    ntsecurity  b''
    reserved2   b'u\x8f\xf6Lx\x05 \xa2\x08\x00\x00\x00\x00\x16*UNREGISTERED vER'
```

(/wp-content/uploads/2019/02/fig12.png)

Figure 12: Header parsing output from acefile.py for the file that produced by the fuzzer in the unexpected path.

These are the first three things that we noticed when we looked at the hex dump and the output from `acefile`:

1. The fuzzer copied parts of the "advert" field to other fields:
    - The content of the compressed file is "SIO", marked in an orange frame in the hex dump. It's part of the advert string "*UNREGISTERED VER**SIO**N*".
    - The filename field contain the string "RED VERSION*" which is part of the advert string "*UNREGISTE**RED VERSION***".
2. The path in the `filename` field was used in the extraction process as an "absolute path" instead of a relative path to the destination folder (the backslash is the root of the drive).
3. The extract file name is "RED VERSION_¶". It seems that the asterisk from the `filename` field was converted to an underscore and the \x14\ (0x14) value represented as "¶" in the extract file name. The other content of the `filename` field is ignored because there is a null char which terminates the string, after the \x14\ (0x14) value.

To find the constraints that caused it to ignore the destination folder and use the `filename` field as an absolute path during the extraction, we did the following attempts, based on our assumptions.

Our first assumption was the first character of the `filename` field (the '\' char) triggers the vulnerability. Unfortunately, after a quick check we found out that this is not the case. After additional checks we arrived at these conclusions:

1. The first char should be a '/' or a '\'.
2. '*' should be included in the `filename` at least once; the location doesn't matter.

Example of a `filename` field that triggers the bug: `\some_folder\some_file*.exe` will be extracted to `C:\some_folder\some_file_.exe` , and the asterisk is converted to an underscore (_).

Now that it worked on our fuzzing harness, it is time to test our crafted archive (e.g. exploit file) file on WinRAR.

Trying the exploit on WinRAR

At first glance, it looked like the exploit worked as expected on WinRAR, because the `sourbe` directory was created in the root of drive `C:\` . However, when we entered the "sourbe" folder (`C:\sourbe`) we noticed that the file was **not** created.

These behaviors raised two questions:

- Why did the harness and WinRAR behave differently?
- Why were the directories that were specified in the exploit file created, and the extracted file was not created?

Why did the harness and WinRAR behave differently?

We expected that the exploit file would behave the same on WinRAR as it behaved in our harness, for the following reasons:

1. The dll (`unacev2.dll`) extracts the files to the destination folder, and not the outer executable (WinRAR or our harness).
2. Our harness mimics WinRAR perfectly when passing parameters / struct members to the dll.

A deeper look showed that we had a false assumption in our second point. Our harness defines 4 callbacks pointers, and our implemented callbacks differ from WinRAR's callbacks. Let's return to our harness implementation.

We mentioned this signature when calling the exported function named `ACEInitDll`.

```
INT __stdcall ACEInitDll(pACEInitDllStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/UNACEFNC.H#LC90) DllData);
```

pACEInitDllStruc (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/UNACEFNC.H#LC90) is a pointer to the sACEInitDLLStruc (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/UNACEFNC.H#LC90) struct. The first member of this struct is tACEGlobalDataStruc (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/STRUCS.H#LC189). This struct has many members, including pointers to callback functions with the following signature:

```
INT (__stdcall *InfoCallbackProc) (pACEInfoCallbackProcStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/STRUCS.H#LC73) Info);

INT (__stdcall *ErrorCallbackProc) (pACEErrorCallbackProcStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/STRUCS.H#LC76) Error);

INT (__stdcall *RequestCallbackProc) (pACERequestCallbackProcStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/STRUCS.H#LC79) Request);

INT (__stdcall *StateCallbackProc) (pACEStateCallbackProcStruc
(https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Include/ACE/includes/STRUCS.H#LC82) State);
```

These callbacks are called by the dll (`unacev2.dll`) during the extraction process.
The callbacks are used as external validators for operations that about to happen, such as the creation of a file, creation of a directory, overwriting a file, etc.
The external callback/validators get information about the operation that's about to occur, for example, file extraction, and returns its decision to the dll.

If the operation is allowed, the following constant is returned to the dll: `ACE_CALLBACK_RETURN_OK` (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Incl if the operation is not allowed by the callback function, it returns the following constant: `ACE_CALLBACK_RETURN_CANCEL` (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Incl and the operation is aborted.

For more information about those callbacks function, see the explanation (https://github.com/FarGroup/FarManager/blob/806c80dff3e182c1c043fad9078490a9bf962456/plugins/newarc.ex/Modules/ace/Incl from the FarManager (https://github.com/FarGroup/FarManager/).

Our harness returned `ACE_CALLBACK_RETURN_OK` for all the callback functions except for the `ErrorCallbackProc`, where it returned `ACE_CALLBACK_RETURN_CANCEL`.

It turns out, WinRAR does validation for the extracted `filename` (after they are extracted and created), and because of those validations in the WinRAR callback's, the creation of the file was aborted. This means that after the file is created, it is deleted by WinRAR.

WinRAR Validators / Callbacks

This is part of the WinRAR callback's validator pseudo-code that prevents the file creation:

```
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            }
```

(/wp-content/uploads/2019/02/fig13.png)

Figure 13: WinRAR validator/callback pseudo-code.

"`SourceFileName`" represents the **relative path** to the file that will be extracted.

The function does the following checks:

1. The first char does not equal "`\`" or "`/`".
2. The File Name doesn't start with the following strings "`..\`" or "`../`" which are gadgets for "Path Traversal".
3. The following "Path Traversal" gadgets does not exist in the string:
    1. "`\..\`"
    2. "`\../`"
    3. "`/../`"
    4. "`/..\`"

The extraction function in `unacv2.dll` calls `StateCallbackProc` in WinRAR, and passes the `filename` field of the ACE format as the relative path to be extracted to.

The relative path is checked by the WinRAR callback's validator. The validators return `ACE_CALLBACK_RETURN_CANCEL` to the dll, (because the `filename` field starts with backslash "\") and the file creation is aborted.

The following string passes to the WinRAR callback's validator:

"\sourbe\RED VERSION_¶"

Note: This is the original filename with fields "\sourbe\RED VERSION*¶". "`unacev2.dll` " replaces the "*" with an underscore.

Why were the folders that were specified in the exploit file created and the extracted file was not created?

Because of a bug in the dll ("`unacev2.dll` "), even if `ACE_CALLBACK_RETURN_CANCEL` returned from the callback, the folders specified in the relative path (`filename` field in ACE archive) will be created by the dll.

The reason for this is that `unacev2.dll` calls the external validator (callback) before the folder creation, but it checks the return value from the callbacks too late – after the creation of the folder. Therefore, it aborts the extraction operation just before writing content to the extracted file, before the call to WriteFile (https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-writefile) API.

It actually creates the extracted file, without writing content to it. It calls to CreateFile (https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) API and then checks the return code from the callback function. If the return code is `ACE_CALLBACK_RETURN_CANCEL`, it actually deletes the file that previously created by the call to CreateFile API.

Side Notes:

- We found a way to bypass the deletion of the file, but it allows us to create empty files only. We can bypass the file deletion by adding ":" to the end of the file, which is treated as Alternate Data Streams (https://blogs.technet.microsoft.com/askcore/2013/03/24/alternate-data-streams-in-ntfs/). If the callback returns `ACE_CALLBACK_RETURN_CANCEL`, dll tries to delete the Alternate Data Stream of the file instead of the file itself.
- There is another filter function in the dll code that **aborts** the extraction operation if the relative path string starts with "****" (slash)**.** This happens in the first extraction stages, before the calls to any other filter function.
  However, by adding **"*"**or "**?**" characters (wildcard characters) to the relative path (filename field) of the compressed file, this check is skipped and the code flow can continue and (partially) trigger the Path Traversal vulnerability. This is why the exploit file which was produced by the fuzzer triggered the bug in our harness. It doesn't trigger the bug in WinRAR because of the callback validator in the WinRAR code.

Summary of Intermediate Findings

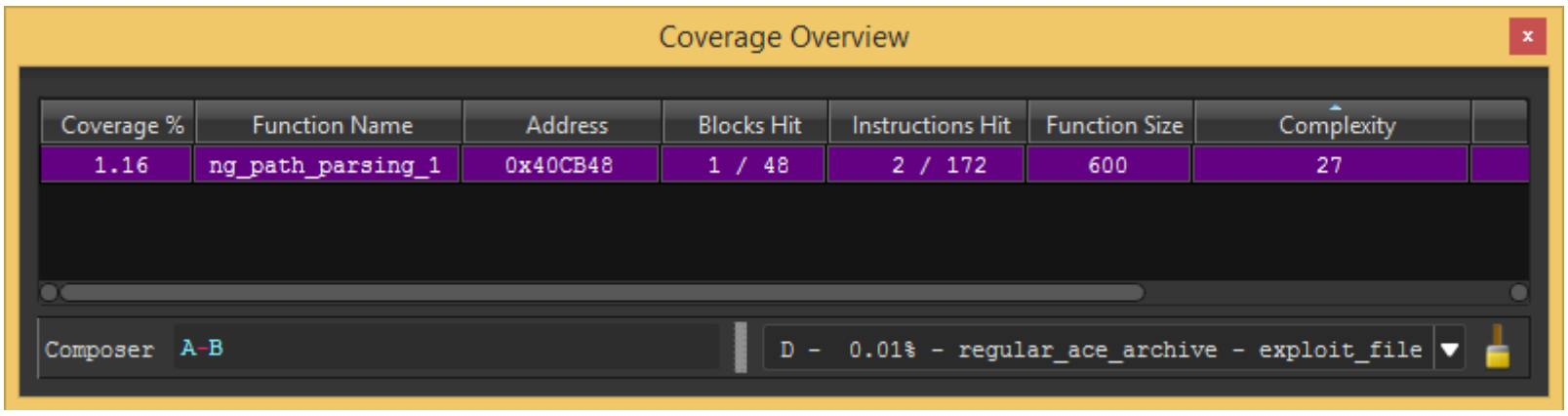
- We found a Path Traversal vulnerability in `unacev2.dll` . It enables our harness to extract the file to an arbitrary path, and completely ignore the destination folder, and treats the extracted file relative path as the full path.
- Two constraints lead to the Path Traversal vulnerability (summarized in previous sections):
  1. The first char should be a '/' or a '\'.
  2. '*' should be included in the `filename` at least once. The location does not matter.
- WinRAR is partially vulnerable to the Path Traversal:
  - `unacev2.dll` doesn't abort the operation after getting the abort code from the WinRAR callback (`ACE_CALLBACK_RETURN_CANCEL`). Due to this delayed check of the return code from WinRAR callback, the directories specified in the exploit file are created.
  - The extracted file is created as well, on the full path specified in the exploit file (without content), but it is deleted right after checking the returned code from the callback (before the call to WriteFile API).
  - We found a way to bypass the deletion of the file, but it allows us to create empty files only.

Finding the Root Cause

At this point, we wanted to figure out why the destination folder is ignored, and the relative path of the archive files (`filename` field) is treated as the full path.

To achieve this goal, we could use static analysis and debugging, but we decided on a much quicker method. We used DynamoRio (https://github.com/DynamoRIO/dynamorio) to record the code coverage in `unacev2.dll` of a regular ACE file and of our exploit file which triggered the bug. We then used the lighthouse (https://github.com/gaasedelen/lighthouse) plugin for IDA and subtracted one coverage path from the other.

These are the results we got:

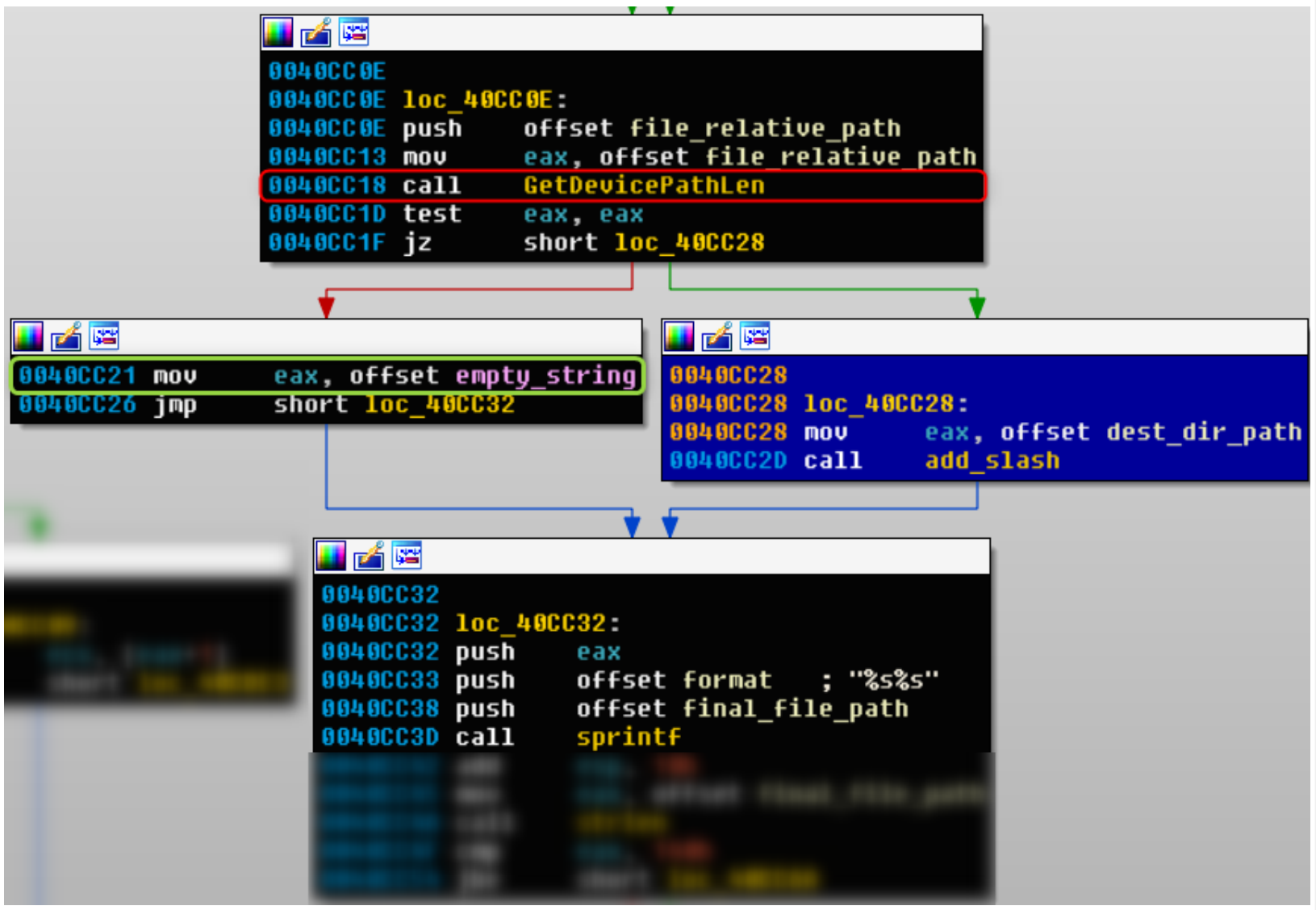


(/wp-content/uploads/2019/02/fig14.png)

Figure 14: Lighthouse's coverage overview window. You can see the coverage subtraction in the "Composer" form, and one result highlighted in purple.

In the "Coverage Overview" window we can see a single result. This means there is only one basic block that was executed in the first attempt (marked in A) and wasn't reached on the second attempt (marked in B).

The Lighthouse plugin marked the background of the diffed basic block in blue, as you can see in the image below.



(/wp-content/uploads/2019/02/fig15.png)

Figure 15: IDA graph view of the main bug in unacev2.dll. Lighthouse marked the background of the diffed basic block in blue.

From the code coverage results, you can understand that the exploit file is **not** going through the diffed basic block (marked in blue), but it takes the opposite basic block (the false condition, marked with a red arrow).

If the code flow goes through the false condition (red arrow) the line that is inside the green frame replaces the destination folder with `""` (empty string), and the later call to `sprintf` function, which concatenates the destination folder to the relative path of the extracted file.

The code flow to the true and false conditions, marked with green and red arrows respectively,
is influenced by the call to the function named `GetDevicePathLen` (inside the red frame).

If the result from the call to `GetDevicePathLen` equals 0, the `sprintf` looks like this:

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

Otherwise:

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

The last `sprintf` is the buggy code that triggers the Path Traversal vulnerability.

This means that the relative path will actually be treated as a fullpath to the file/directory that should be written/created.

Let's look at **GetDevicePathLen** function to get a better understanding of the root cause:

```
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;
}
```

[/wp-content/uploads/2019/02/fig16.png)

Figure 16: GetDevicePathLen code.

The relative path of the extracted file is passed to `GetDevicePathLen`.
It checks if the device or drive name prefix appears in the Path parameter, and returns the length of that string, like this:

- The function returns **3** for this path: **`C:`**`\some_folder\some_file.ext`
- The function returns **1** for this path: `\some_folder\some_file.ext`
- The function returns **15** for this path: **`\\LOCALHOST\C$\`**`some_folder\some_file.ext`
- The function returns **21** for this path: **`\\?\Harddisk0Volume1\`**`some_folder\some_file.ext`
- The function returns **0** for this path: `some_folder\some_file.ext`

If the return value from `GetDevicePathLen` is greater than 0, the relative path of the extracted file will be considered as the full path, because the destination folder is replaced by an empty string during the call to `sprintf`, and this leads to Path Traversal vulnerability.

However, there is a function that "cleans" the relative path of the extract file, by omitting any sequences that are not allowed before the call to `GetDevicePathLen`.

This is a pseudo-code that cleans the path "**`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
}
```

(/wp-content/uploads/2019/02/fig17.png)

Figure 17: Pseudo-code of CleanPath.

The function omits trivial Path Traversal sequences like "**`\..\`**" (it only omits the "**`..\`**" sequence if it is found in the beginning of the path) sequence, and it omits drive sequence like: "**`C:\`**", "**`C:`**", and for an unknown reason, "**`C:\C:`**" as well.

Note that it doesn't care about the first letter; the following sequence will be omitted as well: "**`_:\`**", "**`_:`**", "**`_:\_:`**" (In this case underscore represents any value).

Putting It All Together

To create an exploit file, which causes WinRAR to extract an archived file to an arbitrary path (Path Traversal), extract to the Startup Folder (https://support.microsoft.com/en-us/help/4026268/windows-10-change-startup-apps) (which gains code execution after reboot) instead of to the destination folder.

We should bypass two filter functions to trigger the bug.

To trigger the concatenation of an empty string to the relative path of the compressed file, instead of the destination folder:

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

Instead of:

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

The result from `GetDevicePathLen` function should be greater than 0.
It depends on the content of the relative path ("**file_relative_path**"). If the relative path starts the device path this way:

- **option 1**: `C:\some_folder\some_file.ext`
- **option 2**: `\some_folder\some_file.ext` (The first slash represents the current drive.)

The return value from `GetDevicePathLen` will be greater than 0.
However, there is a filter function in `unacev2.dll` named `CleanPath` (Figure 17) that checks if the relative path starts with **C:** and removes it from the relative path string before the call to `GetDevicePathLen`.

It omits the "**`C:\`**" sequence from the **option 1** string but **doesn't** omit "****" sequence from the **option 2** string.

To overcome this limitation, we can add to **option 1** another "**`C:\`**" sequence which will be omitted by `CleanPath` (Figure 17), and leave the relative path to the string as we wanted with one "**`C:\`**", like:

- **option 1'**: `C:\C:\some_folder\some_file.ext => C:\some_folder\some_file.ext`

However, there is a callback function in WinRAR code (Figure 13), that is used as a validator/filter function. During the extraction process, `unacev2.dll` is called to the callback function that resides in the WinRAR code.

The callback function validates the relative path of the compressed file. If the blacklist sequence is found, the extraction operation will be aborted.

One of the checks that is made by the callback function is for the relative path that starts with "****" (slash).
But it doesn't check for the "**`C:\`**". Therefore, we can use **option 1'** to **exploit the Path Traversal Vulnerability!**

We also found an **SMB attack vector**, which enables it to connect to an arbitrary IP address and create files and folders in arbitrary paths on the SMB server.

Example:
`C:``\\\10.10.10.10\smb_folder_name\some_folder\some_file.ext` **=>** `\\10.10.10.10\smb_folder_name\some_folder\some_file.ext`

Example of a Simple Exploit File

We change the .ace extension to .rar extension, because WinRAR detects the format by the content of the file and not by the extension.

This is the output from `acefile`**:**

```
volume
    filename    C:\Users\nadavgr\Documents\poc.rar
    filesize    141
    headers     MAIN:1 FILE:1 others:0
header
    hdr_crc     0xf760
    hdr_size    49
    hdr_type    0x00        MAIN
    hdr_flags   0x9000      ADVERT!SOLID
    magic       b'**ACE**'
    eversion    20          2.0
    cversion    20          2.0
    host        0x02        Win32
    volume      0
    datetime    0x4e2ea43d  2019-01-14 20:33:58
    reserved1   ba f3 50 11 4e 20 00 00
    advert      b'*UNREGISTERED VERSION*'
    comment     b''
    reserved2   b''
header
    hdr_crc     0x2e58
    hdr_size    62
    hdr_type    0x01        FILE32
    hdr_flags   0x8001      ADDSIZE!SOLID
    packsize    22
    origsize    22
    datetime    0x4e2ea422  2019-01-14 20:33:04
    attribs     0x00000020  ARCHIVE
    crc32       0x8229493d
    comptype    0x00        stored
    compqual    0x03        normal
    params      0x000a
    reserved1   0x4554
    filename    b'c:\\c:\\some_folder\\some_file.txt'
    comment     b''
    ntsecurity  b''
    reserved2   b''
```

(/wp-content/uploads/2019/02/fig18.png)

Figure 18: Header output by acefile.py of the simple exploit file.

We trigger the vulnerability by the crafted string of the `filename` field (in green).

This archive will be extracted to `C:\some_folder\some_file.txt` no matter what the path of the destination folder is.

Creating a Real Exploit

We can gain code execution, by extracting a compressed executable file from the ACE archive to one of the Startup Folders (https://support.microsoft.com/en-us/help/4026268/windows-10-change-startup-apps). Any files that reside in the Startup folders will be executed at boot time.
To craft an ACE archive that extracts its compressed files to the Startup folder seems to be trivial, but it's not.
There are at least 2 Startup folders at the following paths:

1. `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp`
2. `C:\Users\`***`<user name>`***`\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`

The first path of the Startup folder demands high privileges / high integrity level (in case the UAC is on). However, WinRAR runs by default with a medium integrity level.

The second path of the Startup folder demands to know the name of the user.

We can try to overcome it by creating an ACE archive with thousands of crafted compressed files, any one of which contains the path to the Startup folder but with different ***<user name>***, and hope that it will work in our target.

The Most Powerful Vector

We have found a vector which allows us to extract a file to the Startup folder without caring about the ***<user name>***.

By using the following `filename` field in the ACE archive:

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

It is translated to the following path by the `CleanPath` function (Figure 17):

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

Because the `CleanPath` function removes the "**`C:\C:`** " sequence.

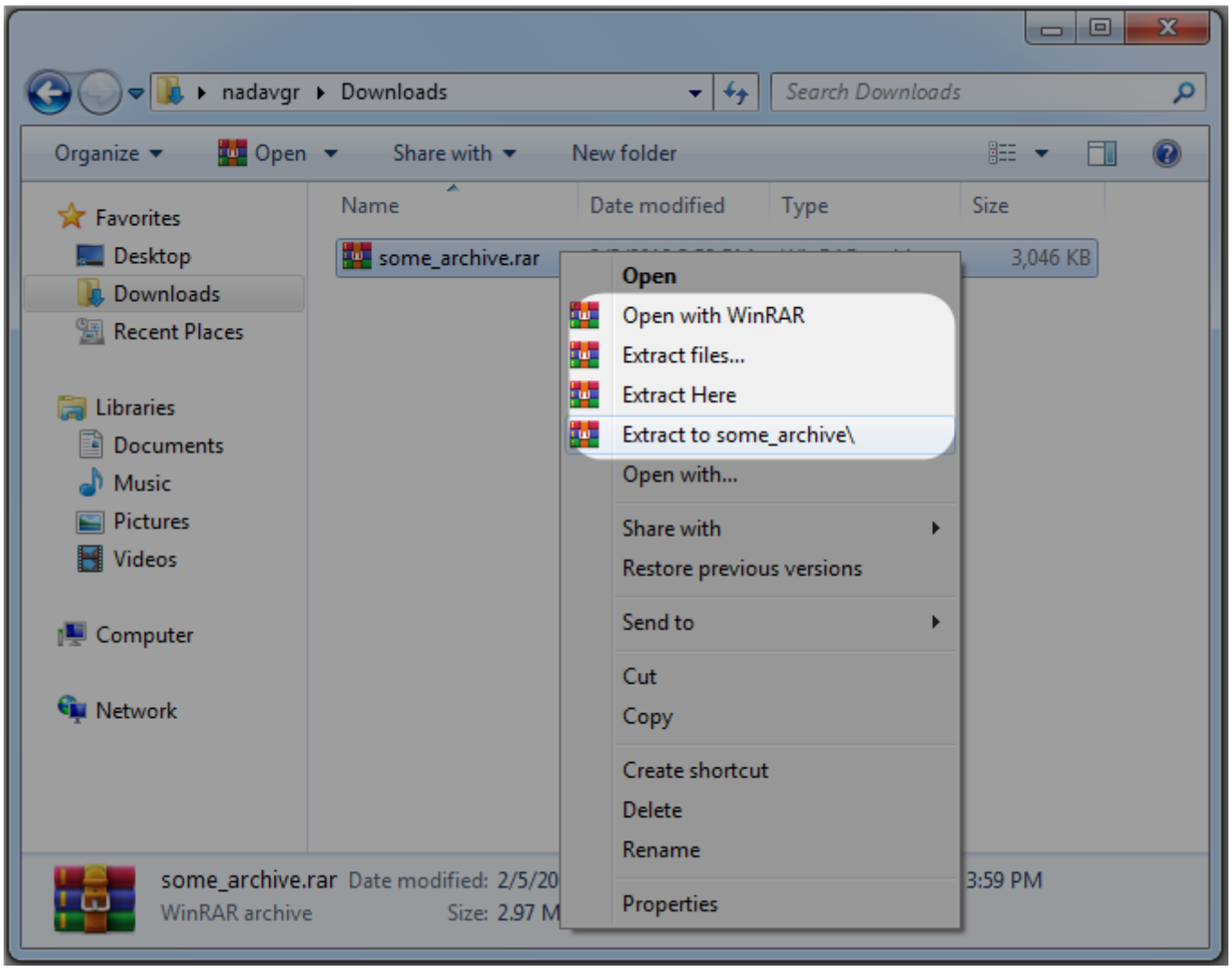
Moreover, this destination folder will be ignored because the `GetDevicePathLen` function (Figure 16) will return 2 for the last "C:" sequence.

Let's analyze the last path:

The sequence "`C:`" is translated by Windows to the "current directory" of the running process. In our case, it's the current path of WinRAR.

If WinRAR is executed from its folder, the "current directory" will be this WinRAR folder: `C:\Program Files\WinRAR`

However, if WinRAR is executed by double clicking on an archive file or by right clicking on "extract" in the archive file, the "current directory" of WinRAR will be the path to the folder that the archive resides in.



(/wp-content/uploads/2019/02/fig19.png)

Figure 19: WinRAR's extract options (WinRAR's shell extension added to write click)

For example, if the archive resides in the user's Downloads folder, the "current directory" of WinRAR will be:

`C:\Users\`***`<user name>`***`\Downloads`

If the archive resides in the Desktop folder, the "current directory" path will be:

`C:\Users\`***`<user name>`***`\Desktop`

To get from the Desktop or Downloads folder to the Startup folder, we should go back one folder "`../`" to the "user folder", and concatenate the relative path to the startup directory: `AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\` to the following sequence: "`C:../`"

This is the end result: `C:../AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\some_file.exe`

Remember that there are 2 checks against path traversal sequences:

- In the `CleanPath` function which skips such sequences.
- In WinRAR's callback function which aborts the extraction operation.

`CleanPath` checks for the following path traversal pattern: "`\..\`"

The WinRAR's callback function checks for the following patterns:

1. "`\..\`"
2. "`\../`"
3. "`/../`"
4. "`/..\`"

Because the first slash or backslash are not part of our sequence "`C:../`", we can bypass the path traversal validation.
However, we can only go back one folder. It's all we need to extract a file to the Startup folder without knowing the user name.

Note: If we want to go back more than one folder, we should concatenate the following sequence "`/../`". For example, "`C:../../`" and the "`/../`" sequence will be caught be the callback validator function and the extraction will be aborted.

Demonstration (POC)

Extracting Code Execution from WinRAR

Side Note

Toward the end of our research, we discovered that `WinACE` created an extraction utility like `unacev2.dll` for linux which is called unace-nonfree (compiled using Watcom compiler). The source code is available.
The source code for Windows (which `unacev2.dll` was built from) is included as well, but it's older than the last version of `unacev2.dll` , and can't be compiled/built for Windows. In addition, some functionality is missing in the source code – for example, the checks in Figure 17 are not included.

However, Figure 16 was taken from the source code.
We also found the Path Traversal bug in the source code. It looks like this:

```
sprintf(BASE_STATE.DestinationFileName, "%s%s",
        BASE_PATHFUNC_GetDevicePathLen(BASE_STATE.CurrentFileName) ?
          "" : BASE_PATHFUNC_AddSlashToEnd(BASE_DIRDATA_Dir2.Dir),
        BASE_STATE.CurrentFileName);
```

(/wp-content/uploads/2019/02/fig20.png)

Figure 20: The path traversal bug in the source code of unace-nonfree

---

CVEs:

CVE-2018-20250, CVE-2018-20251, CVE-2018-20252, CVE-2018-20253.

WinRAR's Response

WinRAR decided to drop UNACEV2.dll from their package, and WinRAR doesn't support ACE format from version number: "5.70 beta 1".

Quote from WinRAR website (https://www.win-rar.com/whatsnew.html?&L=0):

"Nadav Grossman from Check Point Software Technologies informed us about a security vulnerability in UNACEV2.DLL library. Aforementioned vulnerability makes possible to create files in arbitrary folders inside or outside of destination folder when unpacking ACE archives.
WinRAR used this third party library to unpack ACE archives. *UNACEV2.DLL had not been updated since 2005 and we do not have access to its source code.* *So we decided to drop ACE archive format support to protect security of WinRAR users.* We are thankful to Check Point Software Technologies for reporting this issue.“

Check Point's SandBlast Agent Behavioral Guard protect against these threats.

Check Point's IPS blade provides protections against this threat: "RARLAB WinRAR ACE Format Input Validation Remote Code Execution (CVE-2018-20250)"

Many thanks to my colleagues **Eyal Itkin** (@Eyalltkin (https://twitter.com/eyalitkin?lang=en)) and **Omri Herscovici** (@omriher (https://twitter.com/omriher?lang=en)) for their help in this research.

RELATED ARTICLES

ackdoor

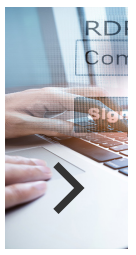
Vol.3 – 2019 Security Report

(//research.checkpoint.com/vol-3-2019-

North Korea Turns Against New Targets?!

Report: Under the Hood of Cyber Crime

(//research.checkpoint.com/2019-



Reverse
RDP Clie