
Checkpoint 1

p7zip

Arnav Nidumolu, Atharva Kale, Pascal von Fintel, Patrick
Negus

2023-04-12

Checkpoint 1

Contents:

- [Overview of the Target](#)
- [Debug Environment](#)
- [Mapping out the Target Code-Base](#)
- [Future Plans](#)

Overview of the Target

p7zip is a fully compliant linux port of the open source *7zip* tool for Windows. It is an utility used to archive and extract various compression formats. It is used primarily in Windows GUI tools as an underlying utility to support their file compression features.

p7zip provides the following features:

1. Several compression algorithms (*lz4*, *zstd*, *Lizard*, etc...)
2. CLI frontend
3. Cryptographic algorithms for archive encryption (*SHA256*, *AES*, *RAR5*, etc...)

Debug Environment

How to build the target

Step 1:

```
1 git clone git@github.com:jinkehan57/p7zip.git
```

We clone [jinkehan57's repo](#), which is a *Linux port* for the 7zip Windows utility. The port is *fully compliant* with the Windows equivalent, and supports all the same formats.

Step 2:

```
1 cp 7zip_gcc_dbg.mak p7zip/CPP/7zip/7zip_gcc.mak
```

We created a custom [Makefile](#) that patches the original build script to include debug flags. We copy the patch into the correct directory.

Step 3:

```
1 cd p7zip/CPP/7zip/Bundles/Alone2 && make -f makefile.gcc && cd -
```

We build the `7zz` tool, which is the primary binary from the project, which supports archiving and extracting the most number of formats.

Step 4:

```
1 PATH=$PATH:$PWD/p7zip/CPP/7zip/Bundles/Alone2/_o/bin
```

For development purposes, we update the current terminal session's `PATH` to include the path to the `7zz` binary.

Experiment with the Target

```
→ 1-checkpoint git:(main) X 7zz -h

7-Zip (z) 22.00 ZS v1.5.2 (x64) : Copyright (c) 1999-2022 Igor Pavlov : 2022-06-15
64-bit locale=en_US.UTF-8 Threads:16

Usage: 7zz <command> [<switches>...] <archive_name> [<file_names>...] [@listfile]

<Commands>
a : Add files to archive
b : Benchmark
d : Delete files from archive
e : Extract files from archive (without using directory names)
h : Calculate hash values for files
i : Show information about supported formats
l : List contents of archive
rn : Rename files in archive
t : Test integrity of archive
u : Update files to archive
x : eXtract files with full paths
```

Figure 1: List of commands

Simple tests

In the `playground` directory, we have some sample files setup for basic tests.

```
1 cd playground
2 7zz a files.zip file1.txt file2.txt
3 7zz e files.zip -ofiles_extracted
```

Target analysis

```
→ bin git:(master) X file 7zz
7zz: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.
2, BuildID[sha1]=02bfdb94fd86cb3d5e02ceaff225f37714a05ad3,
for GNU/Linux 3.2.0, with debug_info, not stripped
→ bin git:(master) X
```

Figure 2: File format

```

+ 390r-debugging-setup git:(main) pwn checksec p7zip/CPP/7zip/Bundles/Alone2/_o/bin/7zz

[*] '/home/lifewhiz/projects/revEng/390r-debugging-setup/p7zip/CPP/7zip/Bundles/Alone2/_o/bin/7zz'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
RUNPATH:   b'$ORIGIN/../lib/7z_addon_codec'
+ 390r-debugging-setup git:(main)

```

Figure 3: checksec mitigations

```

+ 390r-debugging-setup git:(main) x ROPgadget --binary p7zip/CPP/7zip/Bundles/Alone2/_o/bin/7zz
+ 390r-debugging-setup git:(main) x wc -l rop_out.txt
151080 rop_out.txt
+ 390r-debugging-setup git:(main) x cat rop_out.txt | head
Gadgets information
=====
0x000000000043f068 : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0x43ef74
0x0000000000440250 : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0x440154
0x00000000004a902a : adc ah, bh ; cmc ; jmp qword ptr [rsi - 0x70]
0x00000000004a902a : adc ah, bh ; cmc ; jmp qword ptr [rsi - 0x70] ; pop rbx ; ret
0x000000000059626d : adc ah, bh ; dec dword ptr [rax - 0x77] ; ret
0x000000000041f48d : adc ah, bh ; jmp 0x41f35c
0x0000000000485c5d : adc ah, bh ; jmp 0x485ab0
0x0000000000595d1a : adc ah, bh ; jmp 0x594d4a
+ 390r-debugging-setup git:(main) x

```

Figure 4: List of ROP Gadgets

```

+ 390r-debugging-setup git:(main) x readelf -d p7zip/CPP/7zip/Bundles/Alone2/_o/bin/7zz | grep 'NEEDED'
0x0000000000000001 (NEEDED) Shared library: [libzstd.so.1]
0x0000000000000001 (NEEDED) Shared library: [liblz4.so.1]
0x0000000000000001 (NEEDED) Shared library: [libbrotlienc.so.1]
0x0000000000000001 (NEEDED) Shared library: [libbrotlidec.so.1]
0x0000000000000001 (NEEDED) Shared library: [libbrotlicommon.so.1]
0x0000000000000001 (NEEDED) Shared library: [liblzirad.so.1]
0x0000000000000001 (NEEDED) Shared library: [liblz5.so.1]
0x0000000000000001 (NEEDED) Shared library: [libfast-lzma2.so.1]
0x0000000000000001 (NEEDED) Shared library: [liblzhamcomp.so]
0x0000000000000001 (NEEDED) Shared library: [liblzhamdecomp.so]
0x0000000000000001 (NEEDED) Shared library: [liblzhamdll.so]
0x0000000000000001 (NEEDED) Shared library: [libstdc++.so.6]
0x0000000000000001 (NEEDED) Shared library: [libm.so.6]
0x0000000000000001 (NEEDED) Shared library: [libgcc_s.so.1]
0x0000000000000001 (NEEDED) Shared library: [libc.so.6]

```

Figure 5: List of Shared Libraries

```

➔ 390r-debugging-setup git:(main) x one_gadget /usr/lib64/libc.so.6
0x4d170 posix_spawn(rsp+0xc, "/bin/sh", 0, rbx, rsp+0x50, environ)
constraints:
    rsp & 0xf == 0
    rcx == NULL
    rbx == NULL || (u16)[rbx] == NULL

0xf5552 posix_spawn(rsp+0x64, "/bin/sh", [rsp+0x40], 0, rsp+0x70, [rsp+0xf0])
constraints:
    [rsp+0x70] == NULL
    [[rsp+0xf0]] == NULL || [rsp+0xf0] == NULL
    [rsp+0x40] == NULL || (s32)[rsp+0x40+0x4] <= 0

0xf555a posix_spawn(rsp+0x64, "/bin/sh", [rsp+0x40], 0, rsp+0x70, r9)
constraints:
    [rsp+0x70] == NULL
    [r9] == NULL || r9 == NULL
    [rsp+0x40] == NULL || (s32)[rsp+0x40+0x4] <= 0

0xf555f posix_spawn(rsp+0x64, "/bin/sh", rdx, 0, rsp+0x70, r9)
constraints:
    [rsp+0x70] == NULL
    [r9] == NULL || r9 == NULL
    rdx == NULL || (s32)[rdx+0x4] <= 0
➔ 390r-debugging-setup git:(main) x

```

Figure 6: List of One Gadgets

Function call graph

The following can be used to analyze execution of the target and produce graphs. It requires `valgrind` and `kcachegrind` to be installed.

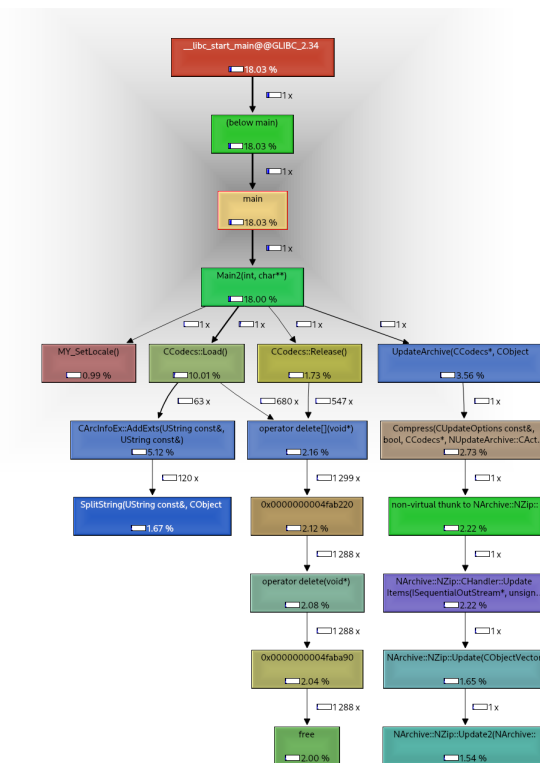
```
1 valgrind --callgrind-out-file=callgrind_vis2 --tool=callgrind 7zz e
  files.zip -ofiles_extracted
```

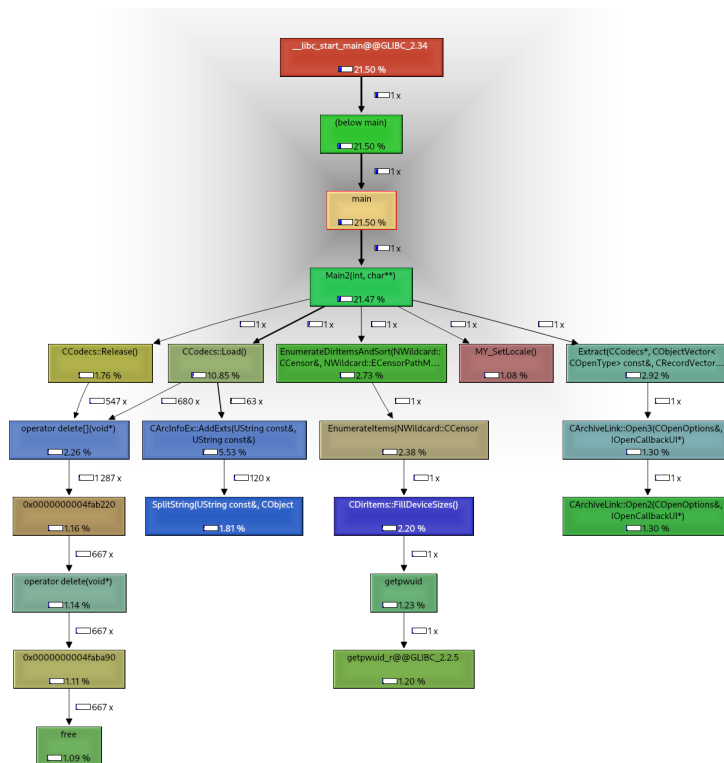
Use the `valgrind` command above to generate a `callgrind_vis2` file.

```
1 kcachegrind callgrind_vis2
```

Use the `kcachegrind` command to visualize the `callgrind_vis2`.

In the next two pages, we find two function call graphs for the `archive` and `extract` subcommands.

Archive Command:**Figure 7:** a subcommand

Extract Command:**Figure 8:** e subcommand

Mapping out the Target Code-Base

The functionality of the console version of this application is straightforward. The binary accepts command line arguments (`main` defined in *MainAr.cpp*), then attempts to pass them to `main2()` in *Main.cpp* (wrapped in try block).

`main2()` handles the bulk of all functionality.

It parses command line arguments beginning at line 733. Argument length is checked, arguments are converted to Unicode and pushed to a string vector.

Arguments are first parsed into the following struct using `parse1()` defined in the *ArchiveCommandLine.cpp*.

```
1 struct CArcCmdLineOptions
2 {
3     bool HelpMode;
4
5     // bool LargePages;
6     bool CaseSensitive_Change;
7     bool CaseSensitive;
8
9     bool IsInTerminal;
10    bool IsStdOutTerminal;
11    bool IsStdErrTerminal;
12    bool StdInMode;
13    bool StdOutMode;
14    bool EnableHeaders;
15
16    bool YesToAll;
17    bool ShowDialog;
18    bool TechMode;
19    bool ShowTime;
20
21    AString ListFields;
22
23    int ConsoleCodePage;
24
25    NWildcard::CCensor Censor;
26
27    CArcCommand Command;
28    UString ArchiveName;
```

First arguments checked are related to showing *help/copyright*, and calls the `ShowCopyRightAndHealth()` function.

Then `parse2()` is called on the options struct.

ArchiveCommandLine.cpp handles a bunch of flags that can be passed, ie. SLP mode (large pages), core affinity, etc. Also contains several other methods for parsing.

Importantly, it defines the formats of arguments. Beginning on line 341, `isFromExtractGroup()` is defined. We see there are `extract`, and `extractFull` flags.

A scanner is defined in *ExtractCallbackConsole.cpp*, and this is presumably used to enumerate files in an archive.

Future Plans

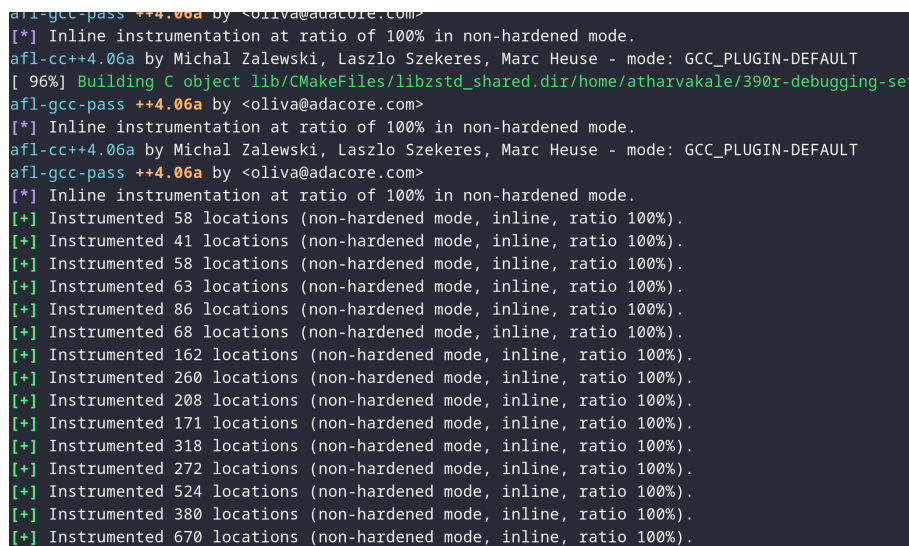
Fuzzing

We plan to use fuzzing as an approach to find bugs in **p7zip**. In specific, mutational fuzzing would be a good fit, where we would use a corpus of *zip* and regular files to test archiving/extracting features of the binary.

To start with, we used the **AFL++** docker container setup:

```
1 docker pull aflplusplus/aflplusplus
2 docker run -ti -v ./src aflplusplus/aflplusplus
```

Then, we recompiled the binary with AFL instrumentation enabled.



```
afl-gcc-pass ++4.06a by <oliva@adacore.com>
[*] Inline instrumentation at ratio of 100% in non-hardened mode.
afl-cc++4.06a by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: GCC_PLUGIN-DEFAULT
[ 96%] Building C object lib/CMakeFiles/libzstd_shared.dir/home/atharvakale/390r-debugging-se
afl-gcc-pass ++4.06a by <oliva@adacore.com>
[*] Inline instrumentation at ratio of 100% in non-hardened mode.
afl-cc++4.06a by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: GCC_PLUGIN-DEFAULT
afl-gcc-pass ++4.06a by <oliva@adacore.com>
[*] Inline instrumentation at ratio of 100% in non-hardened mode.
[+] Instrumented 58 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 41 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 58 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 63 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 86 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 68 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 162 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 260 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 208 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 171 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 318 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 272 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 524 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 380 locations (non-hardened mode, inline, ratio 100%).
[+] Instrumented 670 locations (non-hardened mode, inline, ratio 100%).
```

Figure 9: Compiling with AFL Source Code Instrumentation

We also started a dummy test for archive to see how fuzzing the binary works. Of course, this does not crash the binary, but we got a working template. We need to figure out ways to smarten the input and find coverage through symbolic execution.

