
Checkpoint 1

p7zip

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

2023-04-12

Checkpoint 1

Contents:

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

Overview of the Target

Debug Environment

How to build the target

Pre-requisites:

1. CMake

Step 1:

```
1 git clone git@github.com:jinfaihan57/p7zip.git
```

We clone [jinfaihan57'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

File format

```
→ 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

Mitigations

```

+ 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

ROP Gadgets

```

+ 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

Shared Libraries

```

+ 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: [liblzard.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

One Gadgets

```

➔ 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.

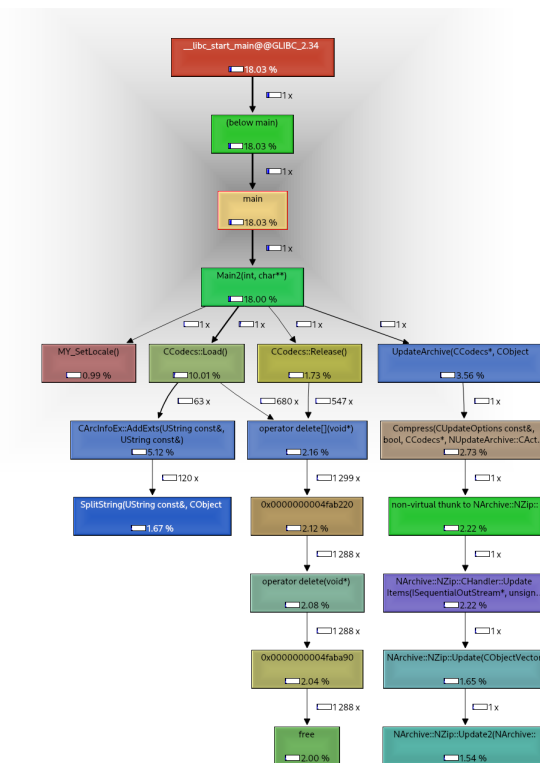
```
1 valgrind --callgrind-out-file=callgrind_vis2 --tool=callgrind 7zz e
  files.zip -ofiles_extracted
```

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:

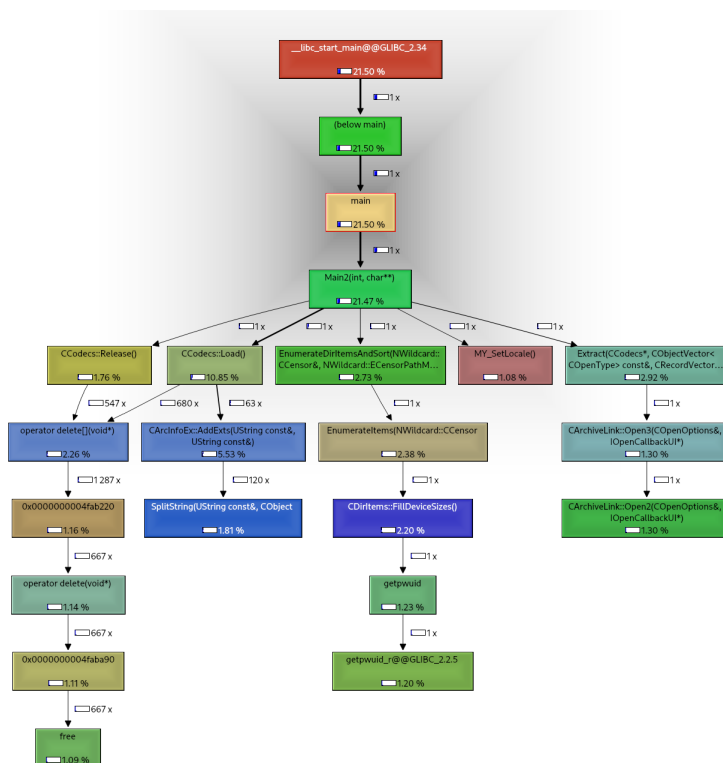


Figure 8: e subcommand

Target Code-Base

TODO

Future Plans