Checkpoint 2

p7zip

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

Checkpoint 2

Contents:

- Checkpoint 2
 - Contents:
 - Static Analysis
 - Dynamic Analysis
 - * Generating a corpus
 - * Experimenting with fuzzing composition flags
 - * Parallel fuzzing
 - * Results

Github Link

https://github.com/atharvakale343/390r-debugging-setup

Dynamic Analysis

Fuzzing

Fuzzing was the main dynamic analysis technique we used against our target p7zip. We mainly fuzzed the extract (e) feature of our binary as the feature uses several decompression algorithms as part of its execution.

We used afl-plus-plus as the primary fuzzing tool.

https://github.com/AFLplusplus/AFLplusplus

Generating a corpus

We took a variety of steps to find a good enough corpus for our fuzzing efforts. The major approach here to was to search online for commonly used corpora. Our main goals here was to find not only .zip format, but also as many different formats possible.

We found a decent corpus at https://github.com/strongcourage/fuzzing-corpus

This included the following formats:

- .zip
- .gzip
- .lrzip
- .jar

We added this as a target to our fuzzing Makefile.

```
1 get-inputs:
      rm -rf in_raw fuzzing-corpus && mkdir in_raw
2
3
      git clone -n --depth=1 --filter=tree:0 git@github.com:strongcourage
          /fuzzing-corpus.git
      cd fuzzing-corpus && git sparse-checkout set --no-cone zip gzip/go-
5
         fuzz lrzip jar && git checkout
      mv fuzzing-corpus/zip/go-fuzz/* in_raw
6
7
      mv fuzzing-corpus/jar/* in_raw
8
      mv fuzzing-corpus/gzip/go-fuzz/* in_raw
      mv fuzzing-corpus/lrzip/* in_raw
```

The next step was to choose only "interesting" inputs from this corpus. This includes small inputs that don't crash that binary immediately.

We used the afl-cmin functionality to minimize the corpus.

```
1 afl-cmin -i in_raw -o in_unique -- $(BIN_AFL) e -y @@
```

Another important minimization step included tmin. This augments each input such that it can be as small as possible without compromising it's ability to mutate and produce coverage in the instrumented target.

Unfortunately, this process takes a long time, and it only completed for us after a day.

```
1 cd in_unique; for i in *; do afl-tmin -i "$$i" -o "../in/$$i" -- ../$(
    BIN_AFL) e -y @@; done
```

The cybersec room servers come in handy here!

Experimenting with fuzzing composition flags

We discovered that it is not enough to fuzz a plain instrumented target with afl-plus-plus. The target binary may not be easily crashed with mutated inputs as p7zip has a robust input error checker. We took to fuzzing with various sanitizers instead to search for harder to find bugs.

We used the following sanitizers on our target:

- ASAN: Address Sanitizer: discovers memory error vulnerabilities such as use-after-free, heap/buffer overflows, initialization order bugs etc.
- MSAN: Memory Sanitizer: mainly used to discover reads to uninitialized memory such as structs etc.
- TSAN: Thread Sanitizer: finds race conditions

```
1 fuzz-afl:
      AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(
          AFL_FUZZ) -M main-afl-$(HOSTNAME) -t 30000 -i in -o out -- $(
          BIN_AFL) e -y @@
3
4 fuzz-afl-asan:
5
      AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(
          AFL_FUZZ) -S variant-afl-asan -t 30000 -i in -o out -- $(
          BIN_AFL_ASAN) e -y @@
6
  fuzz-afl-msan:
7
8
      AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(
          AFL_FUZZ) -S variant-afl-msan -t 30000 -i in -o out -- $(
          BIN_AFL_MSAN) e -y @@
```

Parallel Fuzzing

To start with, our approach was to fuzz the extract command of 7zz. So we found an appropriate corpus and fuzzed with the e command-line argument (along with -y to account for same filenames / avoid user input hangs).

With all different sets of compilation flags that we mentioned previously, we compiled the binaries with AFL instrumentation. Then, to more effectively fuzz, we setup a parallel fuzzing environment in one of the **CyberSec club** VMs.

We added the afl-fuzz commands in a Makefile and followed the official guide for using multiple cores. Below are the commands we utilized. All of our fuzzers shared the same input and output directores to keep track of current fuzzing state.

```
1 AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(AFL_FUZZ)
    -M main-afl-$
2 (HOSTNAME) -t 2000 -i in -o out -- $(BIN_AFL) e -y @@
```

Our main fuzzer used a regular instrumented AFL binary with no other CFLAGS. We used a timeout of 2 seconds to denote a hang (or infinite loops).

```
1 AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(AFL_FUZZ)
    -S variant-af
2 l-asan -t 2000 -i in -o out -- $(BIN_AFL_ASAN) e -y @@
```

Our variant fuzzers utilized binaries compiled with other flags (such as *asan* and *msan*). These had the same timeout as before of 2 seconds.

To keep track of all fuzzers and run them simultaneouly, we used tmux sessions with a separate window for each fuzzer.

Results

We ran the fuzzers using multiple cores for around 2.5 days. We noticed no crashes in most of the variants, with msan being the exception. However, some fuzzers encountered hangs.

```
american fuzzy lop ++4.07a {main-afl-} (...Bundles/Alone2/_o/bin/7zz) [fast]
 process timing
                                                       overall results -
       run time : 2 days, 13 hrs, 15 min, 48 sec
                                                        cycles done : 149
  last new find : 0 days, 0 hrs, 12 min, 9 sec
                                                       corpus count : 9166
last saved crash : none seen yet
                                                      saved crashes : 0
last saved hang : 0 days, 0 hrs, 13 min, 43 sec
                                                        saved hangs: 79
 cycle progress
                                         map coverage
 now processing : 7353.33 (80.2%)
                                           map density : 0.91% / 5.65%
 runs timed out : 0 (0.00%)
                                        count coverage : 5.25 bits/tuple
                                        findings in depth -
- stage progress -
 now trying : splice 1
                                        favored items : 773 (8.43%)
 stage execs : 42/43 (97.67%)
                                        new edges on: 1488 (16.23%)
total execs : 117M
                                        total crashes : 0 (0 saved)
 exec speed: 598.4/sec
                                         total tmouts : 298 (0 saved)
 fuzzing strategy yields
                                                       item geometry
  bit flips : disabled (default, enable with -D)
                                                         levels: 33
 byte flips : disabled (default, enable with -D)
                                                        pending: 140
arithmetics : disabled (default, enable with -D)
                                                       pend fav : 0
 known ints : disabled (default, enable with -D)
                                                      own finds: 8281
 dictionary : n/a
                                                       imported: 198
havoc/splice : 5532/44.1M, 2749/73.6M
                                                      stability: 81.95%
py/custom/rq : unused, unused, unused, unused
    trim/eff : disabled, disabled
                                                               [cpu000:116%]
```

Figure 1: Main AFL Fuzzer

```
american fuzzy lop ++4.07a {variant-afl-asan} (.../Alone2/_o/bin/7zz) [fast]
 process timing
                                                       overall results
       run time : 2 days, 13 hrs, 13 min, 13 sec
                                                        cycles done : 1
  last new find : 0 days, 0 hrs, 10 min, 37 sec
                                                       corpus count : 6737
last saved crash : none seen yet
                                                      saved crashes : 0
last saved hang: 0 days, 1 hrs, 15 min, 58 sec
                                                        saved hangs: 46
 cycle progress
                                         map coverage
 now processing : 5762*0 (85.5%)
                                           map density : 7.40% / 23.60%
 runs timed out : 0 (0.00%)
                                        count coverage : 5.53 bits/tuple
                                        findings in depth
- stage progress -
 now trying : trim 512/512
                                        favored items : 667 (9.90%)
 stage execs : 112/290 (38.62%)
                                         new edges on : 1292 (19.18%)
total execs : 4.64M
                                        total crashes : 0 (0 saved)
 exec speed : 5.56/sec (zzzz...)
                                         total tmouts : 108 (0 saved)
 fuzzing strategy yields
                                                       item geometry
  bit flips : disabled (default, enable with -D)
                                                         levels : 6
 byte flips : disabled (default, enable with -D)
                                                        pending: 3259
arithmetics : disabled (default, enable with -D)
                                                       pend fav : 1
 known ints : disabled (default, enable with -D)
                                                      own finds : 1235
 dictionary : n/a
                                                       imported : 4815
havoc/splice : 422/552k, 813/2.08M
                                                      stability: 69.98%
py/custom/rq : unused, unused, unused, unused
    trim/eff: 8.06%/1.95M, disabled
                                                               [cpu001:100%]
```

Figure 2: Asan Variant Fuzzer

```
american fuzzy lop ++4.07a {variant-afl-msan} (.../Alone2/_o/bin/7zz) [fast]
 process timing
                                                       overall results
       run time : 2 days, 13 hrs, 14 min, 31 sec
                                                        cycles done : 1
  last new find : 0 days, 0 hrs, 2 min, 53 sec
                                                       corpus count : 7427
last saved crash : 0 days, 1 hrs, 13 min, 31 sec
                                                      saved crashes: 978
last saved hang: 0 days, 0 hrs, 36 min, 4 sec
                                                       saved hangs: 94
 cycle progress
                                        map coverage
 now processing : 4889*0 (65.8%)
                                          map density : 1.05% / 5.58%
 runs timed out : 26 (0.35%)
                                        count coverage : 5.37 bits/tuple
                                        findings in depth
- stage progress -
 now trying : trim 8/8
                                        favored items : 722 (9.72%)
 stage execs : 27/90 (30.00%)
                                        new edges on : 1327 (17.87%)
total execs : 4.77M
                                        total crashes : 220k (978 saved)
 exec speed : 30.83/sec (slow!)
                                        total tmouts : 206 (0 saved)
 fuzzing strategy yields
                                                      item geometry
  bit flips : disabled (default, enable with -D)
                                                         levels : 5
 byte flips : disabled (default, enable with -D)
                                                       pending: 2456
arithmetics : disabled (default, enable with -D)
                                                       pend fav : 1
 known ints : disabled (default, enable with -D)
                                                      own finds : 1268
 dictionary : n/a
                                                       imported : 5472
havoc/splice : 465/343k, 1270/1.46M
                                                      stability: 74.96%
py/custom/rq : unused, unused, unused, unused
    trim/eff: 6.55%/2.91M, disabled
                                                               [cpu002:100%]
```

Figure 3: Msan Variant Fuzzer

```
american fuzzy lop ++4.07a {variant-afl-tsan} (.../Alone2/_o/bin/7zz) [fast]
 process timing
                                                      - overall results -
       run time : 2 days, 13 hrs, 11 min, 34 sec
                                                       cycles done : 1
   last new find : 0 days, 0 hrs, 16 min, 36 sec
                                                      corpus count : 7029
last saved crash : none seen yet
                                                      saved crashes : 0
last saved hang : 0 days, 0 hrs, 17 min, 57 sec
                                                       saved hangs : 91
 cycle progress -
                                       map coverage
 now processing : 1058.242 (15.1%)
                                          map density : 6.77% / 24.98%
 runs timed out : 1 (0.01%)
                                       count coverage : 5.52 bits/tuple
 stage progress
                                        findings in depth
                                       favored items : 672 (9.56%)
 now trying : splice 14
 stage execs : 9/12 (75.00%)
                                        new edges on : 1251 (17.80%)
 total execs : 5.40M
                                        total crashes : 0 (0 saved)
 exec speed : 37.03/sec (slow!)
                                        total tmouts : 191 (0 saved)
 fuzzing strategy yields
                                                      item geometry
  bit flips : disabled (default, enable with -D)
                                                        levels : 8
 byte flips : disabled (default, enable with -D)
                                                       pending: 2705
arithmetics : disabled (default, enable with -D)
                                                      pend fav : 0
 known ints : disabled (default, enable with -D)
                                                      own finds : 1435
 dictionary : n/a
                                                      imported: 4907
havoc/splice : 494/775k, 941/2.11M
                                                      stability : 69.15%
py/custom/rq : unused, unused, unused, unused
    trim/eff : 6.48%/2.46M, disabled
                                                               [cpu003:150%]
```

Figure 4: Tsan Variant Fuzzer

We tried running an input from in/hangs to check where an infinite loop could occur. But, all inputs eventually completed while taking longer than 2 seconds, so we concluded that the timeout value was too low. These executions were incorrectly flagged as hangs due to relatively low timeouts. For our next fuzzing attempts, we plan to increase this and make the timout around 30 seconds to account for larger file inputs.

Analyzing msan crashes

As the msan variant was the only one that produced crashes, we compiled a msan binary with debug flags and analyzed the crash.

Figure 5: crash output from msan

As displayed above, the crash occurs due to a use of uninitialized value in FileDir.cpp.

```
else if (S_ISLNK(st.st_mode))
   1086
   1088
   1089
             return true:
   1090
00:0000 rsp <u>0x7fffffffcf60</u> - 0x0
02:0010
              <u>0x7fffffffcf70</u> -- 0x275c09 /* "\t\\'" */
03:0018
              <u>0x7ffffffffff</u> ← 0x1
              0x7ffffffffff680 -- 0x3e8000081b4 -- 0x0
04:0020
              <u>0x7ffffffffff88</u> ← 0x3e8
05:0028
06:0030
              <u>0x7ffffffffff</u> ← 0x0
07:0038
              <u>0x7fffffffcf98</u> ← 0x79 /* 'y' */
                                                                                   - F BACKTRACE 1-
▶ f 0 0x55555575229c NWindows::NFile::NDir::SetFileAttrib_PosixHighDetect(char const*, unsigned int)+268
  f 1  0x555555e2362a CArchiveExtractCallback::SetAttrib()+970
  f 2  0x555555e2a609 CArchiveExtractCallback::SetOperationResult(int)+1817
        0x555555bc962f NArchive::NZip::CHandler::Extract(unsigned int const*, unsigned int, int, IArchiveExtractCallback*)+7775
   f 4 0x555555e80530 Extract(CCodecs*, CObjectVector<COpenType> const&, CRecordVector<int> const&, CObjectVector<UString>&, CObjectVector<U
rd::CCensorNode const&, CExtractOptions const&, IOpenCallbackUI*, IExtractCallbackUI*, IHashCalc*, UString&, CDecompressStat&)+20704
f 5 0x555555e80530 Extract(CCodecs*, CObjectVector<COpenType> const&, CRecordVector<int> const&, CObjectVector<UString>&, CObjectVector<Ustring>&, CObjectVector<Ustring>&, CObjectVector<Ustring>&, CDecompressStat&)+20704
f 6 0x555555f3bc69 Main2(int, char**)+26233
f 7 0x555555f4b575 main+213
   dbg> p st.st_mode
  = 33204
```

Figure 6: gdb analysis for msan crash

From previous error message, we see that msan has flagged st.st_mode as uninitialized. But, looking into **gdb**, this variable seems to be defined, set to 33204. This is because it was called with lstat(path, &st) at the beginning of the function, which initialized all fields of the struct.

From this, we can conclude that msan had incorrectly flagged this an uninitialized and this is a *false positive*. All other msan crashes refer to the same line, so we determined that **msan** is not a good fit for this project, possibly missing out on initializers.

We looked more into clang documentation, which affirmed this belief:

it may introduce false positives and therefore should be used with care

Static Analysis

Codeql

To analyze the code for common C/C++ bugs, we used **codeql** to scan the source code.

We first created a analysis database by providing the make instructions to codeql.

```
1 cd p7zip/CPP/7zip/Bundles/Alone2
2 codeql database create ../../../codeql-playground/analysis-db.
        codeql -l cpp -c "make -B -f makefile.gcc" --overwrite
3 cd -
```

Then, we download *cpp-queries* and tested the produced database against it.

```
codeql pack download codeql/cpp-queriescodeql database analyze analysis-db.codeql --format CSV --output analysis.csv
```

```
Starting evaluation of codeql/cpp-queries/Security/CWE/CWE-704/WcharCharConversion.ql
[42/47 eval 8ms] Evaluation done; writing results to codeql/cpp-queries/Security/CWE/CWE-676/DangerousUseOfCin.bqrs.
Starting evaluation of codeql/cpp-queries/Security/CWE/CWE-732/OpenCallMissingModeArgument.ql.
[43/47 eval 63ms] Evaluation done; writing results to codeql/cpp-queries/Security/CWE/CWE-704/WcharCharConversion.bqrs.
Starting evaluation of codeql/cpp-queries/Security/CWE/CWE-732/UnsafeDaclSecurityDescriptor.ql
[44/47 eval 28ms] Evaluation done; writing results to codeql/cpp-queries/Security/CWE/CWE-732/OpenCallMissingModeArgument.bqrs.
Starting evaluation of codeql/cpp-queries/Summary/LinesOfCode.ql.
[45/47 eval 13ms] Evaluation done; writing results to codeql/cpp-queries/Security/CWE/CWE-732/UnsafeDaclSecurityDescriptor.bqrs.
Starting evaluation of codeql/cpp-queries/Summary/LinesOfUserCode.ql
[46/47 eval 4ms] Evaluation done; writing results to codeql/cpp-queries/Summary/LinesOfCode.bqrs.
[47/47 eval 2.1s] Evaluation done; writing results to codeql/cpp-queries/Summary/LinesOfUserCode.bqrs.
Shutting down query evaluator.
Interpreting results.
Analysis produced the following diagnostic data:
            Diagnostic
                                     | Summary |
 Successfully extracted files | 44 results |
Analysis produced the following metric data:
                              Metric
  Total lines of user written C/C++ code in the database | 4129 |
  Total lines of C/C++ code in the database
                                                                     | 400756 |
  \begin{array}{c} \textbf{codeql-playground} \ \ \texttt{git:}(\textbf{main}) \ \ \textbf{\textit{X}} \ \ \texttt{cat} \ \ \texttt{analysis.csv} \\ \textbf{codeql-playground} \ \ \ \texttt{git:}(\textbf{main}) \ \ \textbf{\textit{X}} \ \boxed{} \end{array}
```

Figure 7: Codesql statistics

But, this did not output any glaring errors, executing without any warnings or useful metrics. We concluded that codeql only utilizes simple checks which would have already been accounted for in the

source code.

CPPCheck

We ran the codebase through the static analysis tool cppcheck, which tagged 1569 warnings and errors. One of the common errors flagged by cppcheck was shiftTooManyBits

```
    390r-debugging-setup\p7zip\CPP\7zip\Archive\7z\p7zln.cpp

    261
    shiftTooManyBits
    758
    error
    Shifting 32-bit value by 32 bits is undefined behaviour

    1546
    shiftTooManyBits
    758
    error
    Shifting 32-bit value by 32 bits is undefined behaviour

    1547
    shiftTooManyBits
    758
    error
    Shifting 32-bit value by 32 bits is undefined behaviour

    1598
    shiftTooManyBits
    758
    error
    Shifting 32-bit value by 32 bits is undefined behaviour

    Shifting 32-bit value by 62 bits is undefined behaviour
```

Unfortunately, when looking at the actual source code, almost all of these errors come from an innocuous function:

The rest, on closer inspection, are also falsely flagged as errors, such as this one:

```
2594 shiftTooManyBits 758 error Shifting 32-bit value by 63 bits is undefined behaviour

2594 if (node. FileSize ≥ ((UInt64)1 ≪ 63))

2595 return S_FALSE;
```

A more promising error seems to be a possible null pointer exception:

```
| 390r-debugging-setup\p7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CPP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\CP\7zip\
```

```
37
     static bool IsSimpleAsciiString(const wchar_t *s)
38
        for (;;)
40
41
          wchar_t c = *s++;
42
          if (c = 0)
            return true;
43
          if (c < 0×20 || c > 0×7F)
            return false;
45
        }
47
```

This function is only called once, in the same file at line 415:

```
CMyComPtr<ICryptoGetTextPassword2> getTextPassword;
          CMyComPtr<IArchiveUpdateCallback> udateCallBack2(callback);
          udateCallBack2.QueryInterface(IID_ICryptoGetTextPassword2, &getTextPassword);
        CCompressionMethodMode options;
        (CBaseProps &)options = _props;
        options._dataSizeReduce = largestSize;
        options._dataSizeReduceDefined = largestSizeDefined;
        options.PasswordIsDefined = false;
        options.Password.Wipe_and_Empty();
        if (getTextPassword)
          CMyComBSTR_Wipe password;
406
          Int32 passwordIsDefined;
          RINOK(getTextPassword→CryptoGetTextPassword2(&passwordIsDefined, &password));
          options.PasswordIsDefined = IntToBool(passwordIsDefined);
          if (options.PasswordIsDefined)
            if (!m_ForceAesMode)
              options.IsAesMode = thereAreAesUpdates;
            if (!IsSimpleAsciiString(password))
              return E_INVALIDARG;
```

It looks like password gets populated in CryptoGetTexPassword2, looking at that function, and the subsequent call to StringToBstr, it unfortunately looks like the nullpointer is properly checked for.