Checkpoint 2

p7zip

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

2023-04-06

Checkpoint 2

Contents:

Github Link

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

Static Analysis

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.

```
get-inputs:
    rm -rf in_raw fuzzing-corpus && mkdir in_raw

    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-
        fuzz lrzip jar && git checkout
    mv fuzzing-corpus/zip/go-fuzz/* in_raw
    mv fuzzing-corpus/jar/* in_raw
    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.

```
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.

```
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

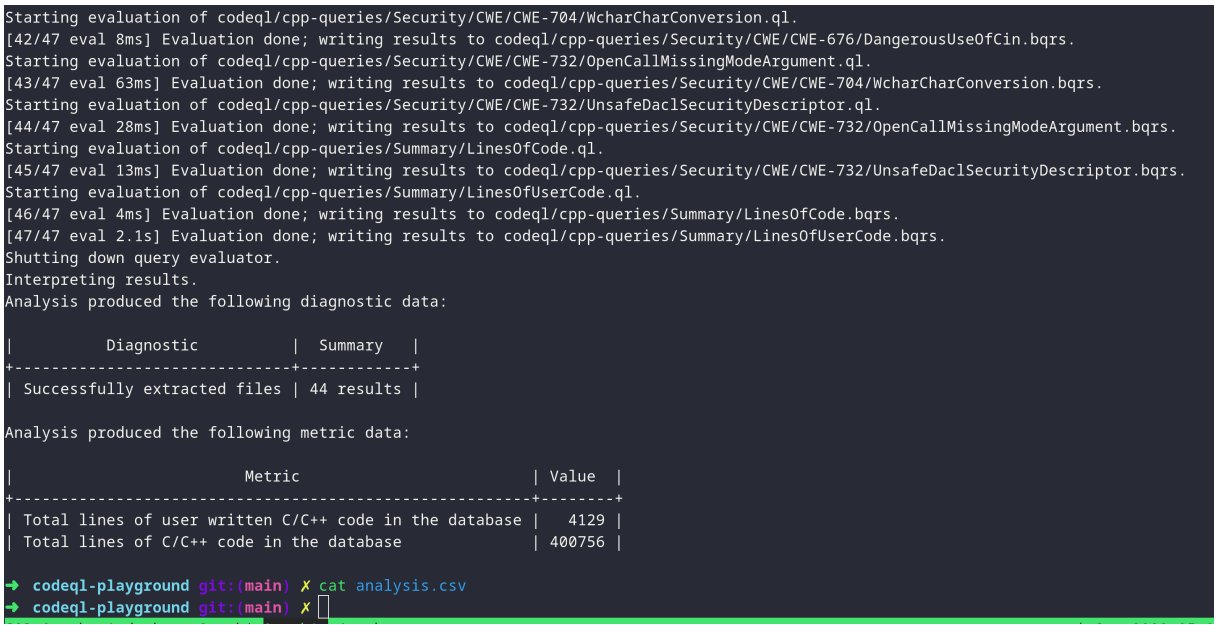
We discovered that it is not enough to fuzz a plain instrumented target with `afl-plus-plus`. The target binary may not be easily crashed with mutated inputs as `p7zip` has a robust input error checker. We took to fuzzing with various sanitizers instead to search for harder to find bugs.

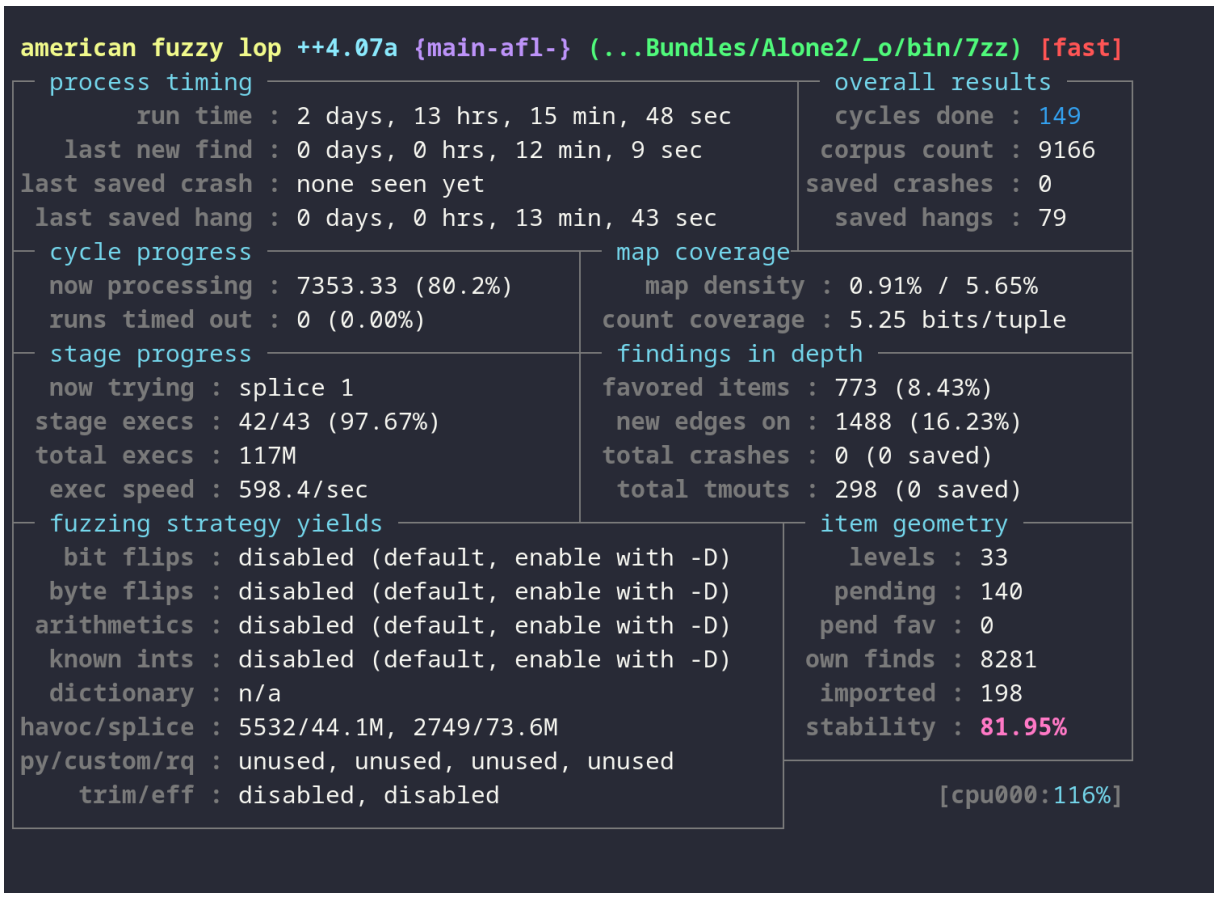
We used the following sanitizers on our target:

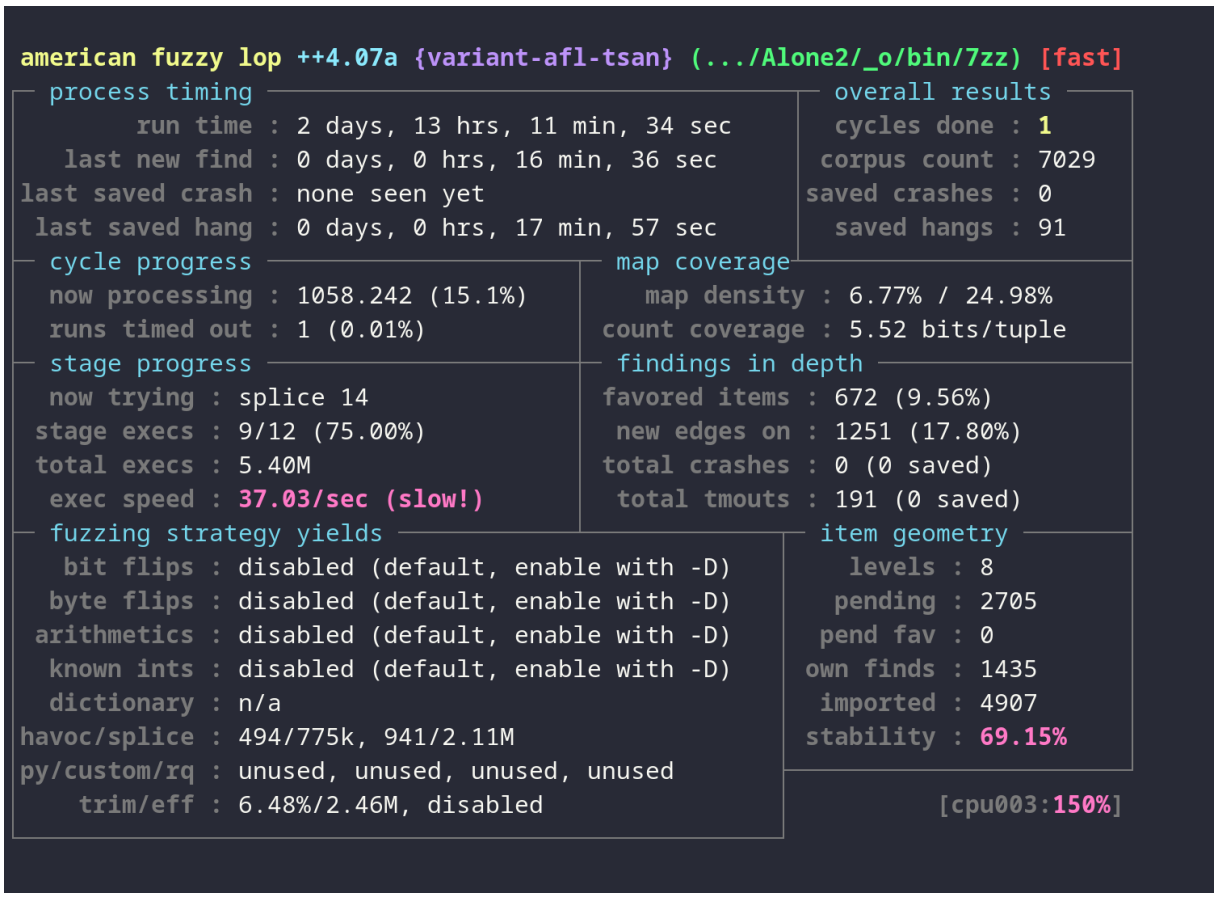
- ASAN: Address Sanitizer: discovers memory error vulnerabilities such as use-after-free, heap/buffer overflows, initialization order bugs etc.
- MSAN: Memory Sanitizer: mainly used to discover reads to uninitialized memory such as structs etc.
- TSAN: Thread Sanitizer: finds race conditions

```
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 @@

fuzz-afl-asan:
    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 @@

fuzz-afl-msan:
    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 @@

fuzz-afl-tsan:
```

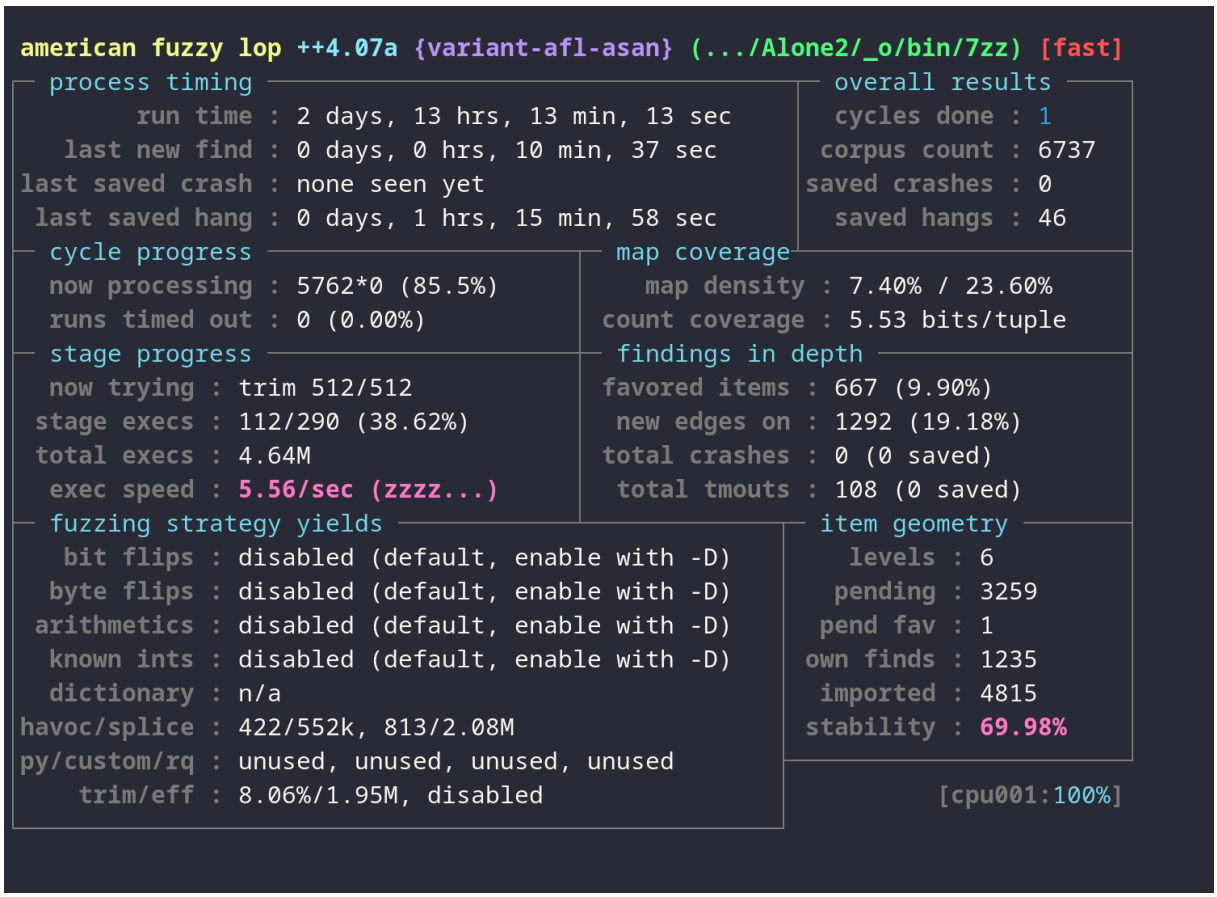
```
11      AFL_SKIP_CPUFREQ=1 AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 $(
            AFL_FUZZ) -S variant-afl-tsan -t 30000 -i in -o out -- $(
            BIN_AFL_TSAN) e -y @@
```




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


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


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



Static Analysis

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\7zIn.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 62 bits is undefined behaviour

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

```
#define GetUi64(p) (GetUi32(p) | ((UInt64)GetUi32((const Byte *)(p)) + 4) << 32))
```

```
#define GetUi32(p) ( \
            ((const Byte *)(p))[0]         | \
    ((UInt32)((const Byte *)(p))[1] <<  8) | \
    ((UInt32)((const Byte *)(p))[2] << 16) | \
    ((UInt32)((const Byte *)(p))[3] << 24))
```

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\Archive\Zip\ZipHandlerOut.cpp

41	nullPointerRedundantCheck	476	warning	Either the condition 'password' is redundant or there is possible null pointer dereference: s++.
41	nullPointerArithmeticRedundantCheck	682	warning	Either the condition 'password' is redundant or there is pointer arithmetic with NULL pointer.

```
37    static bool IsSimpleAsciiString(const wchar_t *s)
38    {
39      for (;;)
40      {
41        wchar_t c = *s++;
42        if (c == 0)
43          return true;
44        if (c < 0x20 || c > 0x7F)
45          return false;
46      }
47    }
```

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

```
392     CMyComPtr<ICryptoGetTextPassword2> getTextPassword;
393     {
394       CMyComPtr<IArchiveUpdateCallback> udateCallBack2(callback);
395       udateCallBack2.QueryInterface(IID_ICryptoGetTextPassword2, &getTextPassword);
396     }
397     CCompressionMethodMode options;
398     (CBaseProps &)options = _props;
399     options._dataSizeReduce = largestSize;
400     options._dataSizeReduceDefined = largestSizeDefined;
401
402     options.PasswordIsDefined = false;
403     options.Password.Wipe_and_Empty();
404     if (getTextPassword)
405     {
406       CMyComBSTR_Wipe password;
407       Int32 passwordIsDefined;
408       RINOK(getTextPassword->CryptoGetTextPassword2(&passwordIsDefined, &password));
409       options.PasswordIsDefined = IntToBool(passwordIsDefined);
410       if (options.PasswordIsDefined)
411       {
412         if (!m_ForceAesMode)
413           options.IsAesMode = thereAreAesUpdates;
414
415         if (!IsSimpleAsciiString(password))
416           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.

```
97  STDMETHODIMP CUpdateCallback100Imp::CryptoGetTextPassword2(Int32 *passwordIsDefined, BSTR *password)
98  {
99    *password = NULL;
100   *passwordIsDefined = BoolToInt(PasswordIsDefined);
101   if (!PasswordIsDefined)
102     return S_OK;
103   return StringToBstr(Password, password);
104 }
```

```
77  inline HRESULT StringToBstr(LPCOLESTR src, BSTR *bstr)
78  {
79    *bstr = ::SysAllocString(src);
80    return (*bstr) ? S_OK : E_OUTOFMEMORY;
81  }
```