p7zip

a port of 7zip for posix

Arnav Nidumolu
Atharva Kale
Pascal von Fintel
Patrick Negus

Agenda

01 Overview of Target

02 Corpus for Dynamic analysis

03 Fuzzing with AFL++

04 Static analysis with CPPCheck & CodeQL

Overview of Target

Target

- p7zip is an open-source POSIX port of the popular Windows utility 7zip
- Archive and extract various compression formats
  - Implements several compression algorithms (codecs)
- Hosts a CLI frontend
- Used in almost all GUI ZIP programs in Windows
- Cryptographic algorithms for archive encryption

```
➜  390r-debugging-setup git:(main) ✗ 7zz -h

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

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

<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
```

Debug environment

All build instructions are in our Github Repo:
https://github.com/atharvakale343/p7zip-390r

We have a Makefile that supports the following build targets:

- default (as-is)
- debug (-g2 -O0 CFLAGS)
- AFL instrumentation (afl)
  - ASAN
  - MSAN
  - UBSAN
  - CFISAN
  - TSAN

```
$ make afl-msan
```

Dynamic Analysis of Target

Fuzzing

- Popular technique against binary targets
- Fuzzed p7zip's extract (e) subcommand
  - Used afl-plus-plus
- Requires a varied corpus
  - Files across several compression formats
  - Found a list on GitHub

```
$ make get-inputs
```

Corpus Minimization

- Significantly increased coverage of our fuzzing efforts
- `afl-cmin`
  - Ensures only "interesting" inputs are considered
  - Considers coverage
- `afl-tmin`
  - Strips the input to make it lightweight
  - Retains similar state/output

```
→  fuzzing-work2 git:(main) make minimize-afl
rm -rf in_unique in
afl-cmin -i in_raw -o in_unique -- ../../7zz_afl/CPP/7zip/Bundles/Alone2/_o/bin/7
corpus minimization tool for afl++ (awk version)

[+] Setting AFL_MAP_SIZE=65536
[*] Testing the target binary...
[!] WARNING: Setting an execution timeout of 120 seconds ('none' is not allowed)
[+] OK, 2023 tuples recorded.
[*] Obtaining traces for 687 input files in 'in_raw'.
    Processing 687 files (forkserver mode)...
[!] WARNING: Setting an execution timeout of 120 seconds ('none' is not allowed)
[*] Processing traces for input files in 'in_raw'.
    Processing file 687/687
    Processing tuple 12214/12214 with count 687...[+] Found 12214 unique tuples a
[+] Narrowed down to 163 files, saved in 'in_unique'.
mkdir in
cd in_unique; for i in *; do afl-tmin -i "$i" -o "../in/$i" -- ../../../7zz_afl/C
afl-tmin++4.07a by Michal Zalewski
```

```
$ make minimize
```

Fuzzing with AFL++

Fuzzing with only AFL

- Discovered not sufficient to just fuzz a vanilla AFL binary
- No crashes or hangs
- Next Step: Fuzzing composition flags
  - sanitizers to find non-crashing bugs

```
american fuzzy lop ++4.07a {main-afl-} (...Bundles/Alone2/_o/bin/7zz) [fast]
┌─ process timing ─────────────────────┬─ overall results ─────┐
│        run time : 2 days, 20 hrs, 37 min, 24 sec │  cycles done : 126    │
│   last new find : 0 days, 0 hrs, 17 min, 19 sec  │ corpus count : 10.7k  │
│last saved crash : none seen yet                  │saved crashes : 0      │
│ last saved hang : none seen yet                  │  saved hangs : 0      │
├─ cycle progress ─────────────────────┼─ map coverage ──────────┤
│  now processing : 4557.43 (42.8%)    │   map density : 1.85% / 6.44%  │
│  runs timed out : 0 (0.00%)          │count coverage : 5.07 bits/tuple│
├─ stage progress ─────────────────────┼─ findings in depth ─────┤
│  now trying : splice 3               │favored items : 968 (9.09%)     │
│ stage execs : 60/129 (46.51%)        │ new edges on : 1718 (16.13%)   │
│ total execs : 124M                   │total crashes : 0 (0 saved)     │
│  exec speed : 552.3/sec              │ total tmouts : 0 (0 saved)     │
├─ fuzzing strategy yields ────────────┴─ item geometry ──────┤
│   bit flips : disabled (default, enable with -D)  │    levels : 44        │
│  byte flips : disabled (default, enable with -D)  │   pending : 1096      │
│ arithmetics : disabled (default, enable with -D)  │  pend fav : 0         │
│  known ints : disabled (default, enable with -D)  │own finds : 10.4k      │
│  dictionary : n/a                                 │ imported : 80         │
│havoc/splice : 6375/47.4M, 4033/76.6M              │stability : 87.62%     │
│py/custom/rq : unused, unused, unused, unused      │                       │
│    trim/eff : disabled, disabled                  │          [cpu00       │
└──────────────────────────────────────────────────┴──────────────┘
```

Fuzzing Composition Flags

We used the following sanitizers on our target:

1. ASAN: Address Sanitizer
    - Discover memory error vulnerabilities such as use-after-free, heap/buffer overflows, initialization order bugs, etc.
2. MSAN: Memory Sanitizer
    - Discover reads to uninitialized memory such as structs
3. TSAN: Thread Sanitizer
    - Find race conditions

```
$ make fuzz-afl-<sanitizer>
```

Parallel Fuzzing

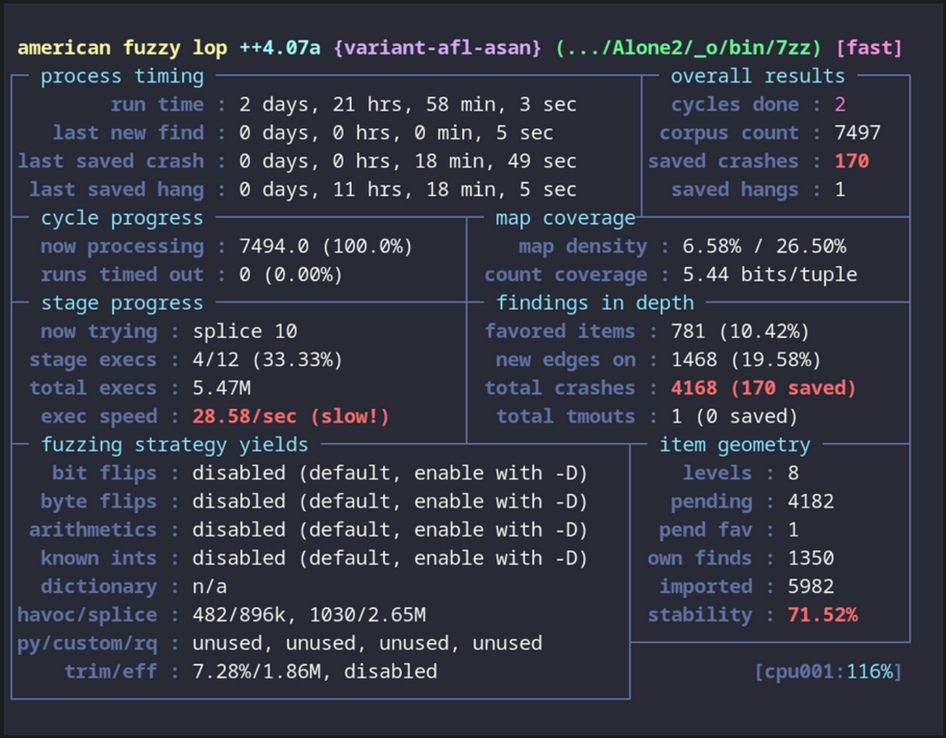
- Efficient to have one fuzzer running of each type
- Main fuzzer supported by several variant fuzzers
- All fuzzers share same output directory

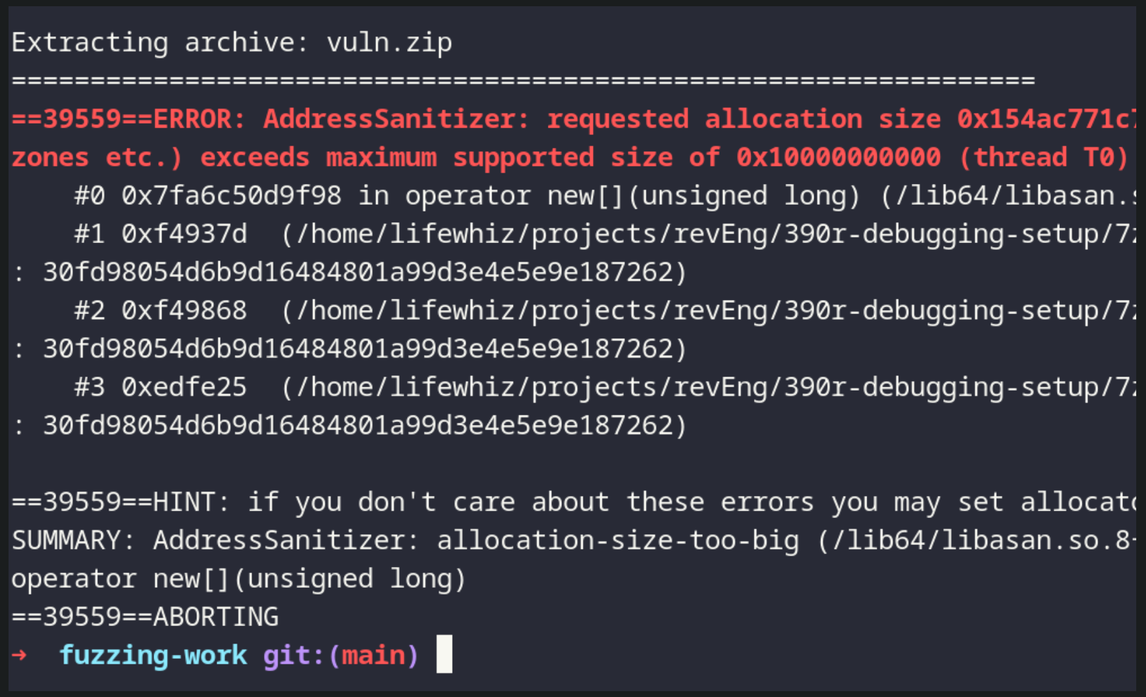
```
Summary stats
=============

       Fuzzers alive : 4
      Total run time : 5 days, 17 hours
         Total execs : 129 millions
    Cumulative speed : 568 execs/sec
       Average speed : 142 execs/sec
       Pending items : 111 faves, 5601 total
  Pending per fuzzer : 27 faves, 1400 total (on average)
       Crashes saved : 167
Cycles without finds : 0/0/0/0
  Time without finds : 24 seconds
```

Cybersec Room servers ftw

Fuzzing results

ASAN-variant





- Only this fuzzer found crashes
- Hang is a false-positive (unable to reproduce)

- Allocation exceeded capacity
- Potential bug

Analyzing the error

```
HRESULT CUnpacker::UnpackData(IInStream *inStream,
    const CResource &resource, const CHeader &header,
    const CDatabase *db,
    CByteBuffer &buf, Byte *digest)
{
  // if (resource.IsSolid()) return E_NOTIMPL;

  UInt64 unpackSize64 = resource.UnpackSize;
  if (db)
    unpackSize64 = db->Get_UnpackSize_of_Resource(resource);

  size_t size = (size_t)unpackSize64;
  if (size != unpackSize64)
    return E_OUTOFMEMORY;

  buf.Alloc(size);

  CBufPtrSeqOutStream *outStreamSpec = new CBufPtrSeqOutStream();
  CMyComPtr<ISequentialOutStream> outStream = outStreamSpec;
  outStreamSpec->Init((Byte *)buf, size);

  return Unpack(inStream, resource, header, db, outStream, NULL, digest);
}
```

- UnpackSize is passed into Alloc

```
   46
   47   void Alloc(size_t size)
   48   {
 ▶ 49     if (size != _size)
   50     {
   51       Free();
   52       if (size != 0)
   53       {
   54         _items = new T[size];
[ STACK ]
00:0000 rsp 0x7fffffffbb70 ◂— 0x154ac771c7ffffff
01:0008     0x7fffffffbb78 —▸ 0x7fffffffbd00 ◂— 0x0
02:0010 rbp 0x7fffffffbb80 —▸ 0x7fffffffbbf0 —▸ 0x7fffffffbcb0 —▸ 0x7fffffffbed
03:0018     0x7fffffffbb88 —▸ 0x578e16 ◂— mov edi, 0x28
04:0020     0x7fffffffbb90 —▸ 0x7fffffffbd00 ◂— 0x0
05:0028     0x7fffffffbb98 —▸ 0x76d2a8 ◂— 0x0
06:0030     0x7fffffffbba0 —▸ 0x7fffffffbd80 ◂— 0x4034b5000010ae7
07:0038     0x7fffffffbba8 —▸ 0x7fffffffbdd0 ◂— 0x800000000
[ BACKTRACE ]
 ▶ f 0         0x45bc1c
   f 1         0x578e16
   f 2         0x57abcd
   f 3         0x56ba3f
   f 4         0x64612e
   f 5         0x6495ab
   f 6         0x64a3ab
   f 7         0x64a9f1 CArc::OpenStreamOrFile(COpenOptions&)+363

pwndbg> p/x size
$1 = 0x154ac771c7ffffff
pwndbg>
```

- new (internally malloc) is called with a huge size argument

Looking at the file

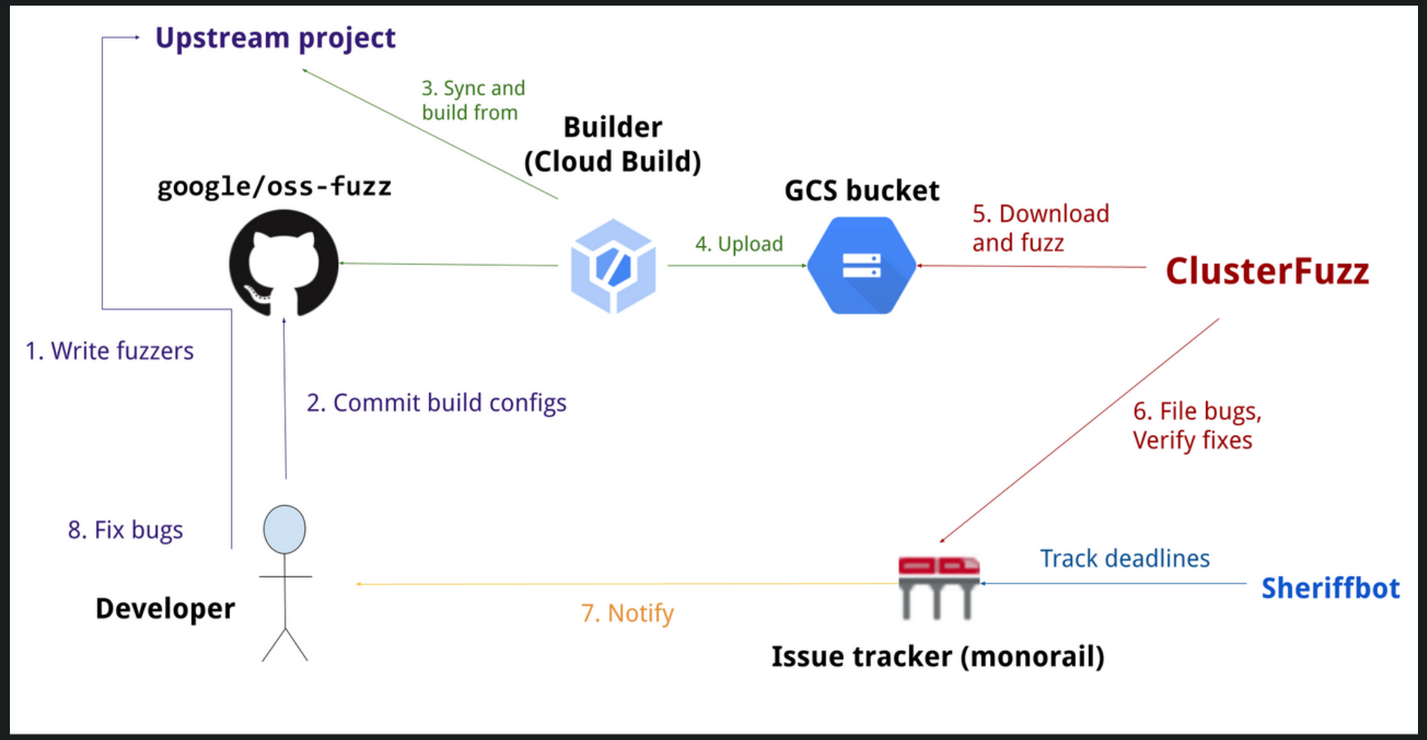
```
→  fuzzing-work git:(main) xxd -p vuln.zip | head | grep ffffffc771c74a15
0300ffffffffc771c74a151432000003f00300000040030004000300000
→  fuzzing-work git:(main)
```

- size is extracted from from offset in the file header
- Potential bug if combined with static/taint analysis
  - find arithmetic operations used with this malloc argument

Automated Fuzzing with OSS-Fuzz

Continuous Fuzzing

- A pull request mentions fuzzing p7zip with CI/CD
- Continually fuzzing open-source projects and reporting live crashes
- "Fuzzing techniques with scalable, distributed execution"



Static Analysis with CppCheck, CodeQL, Flawfinder

Static Analysis

- Analyzed codebase using CppCheck, CodeQL, Flawfinder
  - CppCheck relies on multiple integrated tools for analyzing source; focuses on detecting undefined behavior
  - CodeQL abstracts the source to a QL-language IR, which can then be queried
  - Flawfinder is a syntatic analysis engine that scans for vulnerable code patterns
- Motivations:
  - Run codebase through multiple tools in order to find bugs
  - Static analysis tools are significantly more effective at finding vulnerabilities when combined*

*CPPCheck/Flawfinder in particular when run alone struggle to identify vulnerabilities, according to Lip et al. 2022 empirical study (preprint)

Static Analysis with CppCheck

- CppCheck tagged a large number of errors, but most were false positives associated with a macro

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

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

Static Analysis with CppCheck

- Potential nullpointer, but also unfortunately a false positive:

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  }
```

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

Static Analysis with CodeQL

- Tested against standard queries *cpp-queries*
- Nothing interesting produced
- Default queries likely insufficient for probing our target

```
|          Diagnostic           |  Summary   |
+-------------------------------+------------+
| Successfully extracted files | 44 results |

Analysis produced the following metric data:

|                          Metric                          | Value  |
+----------------------------------------------------------+--------+
| Total lines of user written C/C++ code in the database |   4129 |
| Total lines of C/C++ code in the database              | 400756 |

➜  codeql-playground git:(main) ✗ cat analysis.csv
➜  codeql-playground git:(main) ✗
```

Static Analysis with CodeQL

- Crafted a CodeQL query to find calls to malloc where the input contained arithmetic operators, to look for overflows and other allocation bugs
- First pass of local flow had 82 results

```
import cpp
import semmle.code.cpp.dataflow.DataFlow

from Function malloc, FunctionCall fc, Expr src
where malloc.hasGlobalName("malloc")
  and fc.getTarget() = malloc
  and DataFlow::localFlow(DataFlow::exprNode(src), DataFlow::exprNode(fc.getArgument(0)))
  and src.toString().regexpMatch(".*([\\+\\/\\-\\=\\*]+).*")
select src
```

Example Hit ->

```
466  FSE_CTable* FSE_createCTable (unsigned maxSymbolValue, unsigned tableLog)
467  {
468      size_t size;
469      if (tableLog > FSE_TABLELOG_ABSOLUTE_MAX) tableLog = FSE_TABLELOG_ABSOLUTE_MAX;
470      size = FSE_CTABLE_SIZE_U32 (tableLog, maxSymbolValue) * sizeof(U32);
471      return (FSE_CTable*)malloc(size);
472  }
```

Static Analysis with CodeQL

- Unfortunately, our target (or at least our specific target) is the cpp version of p7zip, and the new/new[] operator replaces malloc for object initialization
- However, the wrapper function Alloc() takes in a size parameter
- So, rewrite query to find all instances where arithmetic operations control the size parameter:

```
import cpp
import NewDelete
import semmle.code.cpp.Print


from
  Function alloc, FunctionCall fc, Expr src
where alloc.getName() = "Alloc"
  and fc.getTarget() = alloc
  and DataFlow::localFlow(DataFlow::exprNode(src), DataFlow::exprNode(fc.getArgument(0))
  and src.toString().regexpMatch(".*([\\+\\/\\-\\=\\*]+)*.")

select src
```

#select ▾ 287 results

#	src
1	call to GetRem
2	rem
3	... + ...
4	call to GetFolderUnpackSize
5	unpackSize64
6	unpackSize
7	unpackSize
8	1
9	NumFolders
10	NumFolders
11	call to ReadNum

287 results!

Static Analysis with CodeQL

- Use global query to find all calls to UnpackData(), to bug hunt crash found by fuzzer

```
class EnvironmentToFileConfiguration extends DataFlow::Configuration {
  Quick Evaluation: EnvironmentToFileConfiguration
  EnvironmentToFileConfiguration() { this = "EnvironmentToFileConfiguration" }

  Quick Evaluation: isSource
  override predicate isSource(DataFlow::Node source) {
    exists(Function getenv |
      source.asIndirectExpr(1).(FunctionCall).getTarget() = getenv
    )
  }

  Quick Evaluation: isSink
  override predicate isSink(DataFlow::Node sink) {
    exists(FunctionCall fc |
      sink.asIndirectExpr(1) = fc.getArgument(0) and
      fc.getTarget().getName() = "UnpackData"
    )
  }
}

from
  Expr anyfunc, Expr unpackData, EnvironmentToFileConfiguration config, DataFlow::Node source
  DataFlow::Node sink
where
  source.asIndirectExpr(1) = anyfunc and
  sink.asIndirectExpr(1) = unpackData and
  config.hasFlow(source, sink)
select unpackData, "This 'unpackData' uses data from $@.", anyfunc, "call to 'anyfunc'"
```

#	unpackData	[1]	anyfunc	[3]
1	inStream	This 'unpackData' uses data from $@.	call to operator IInStream *	call to 'anyfunc'
2	call to operator IInStream *	This 'unpackData' uses data from $@.	call to operator IInStream *	call to 'anyfunc'
3	inStream	This 'unpackData' uses data from $@.	call to operator IInStream *	call to 'anyfunc'
4	inStream	This 'unpackData' uses data from $@.	call to operator IInStream *	call to 'anyfunc'

Static Analysis with CodeQL

- As stated, issue occurs when h.XmlResource.unpackSize is extremely large
- Thus, a corrupt XmlResource field in the zip header would trigger CPP error handling for the new keyword
- Not a particularly serious issue
- Could possibly solve by checking sizes of CResource fields

```
HRESULT CDatabase::OpenXml(IInStream *inStream, const CHeader &h, CByteBuffer &xml)
{
  CUnpacker unpacker;
  return unpacker.UnpackData(inStream, h.XmlResource, h, this, xml, NULL);
}
```

```
struct CResource
{
  UInt64 PackSize;
  UInt64 Offset;
  UInt64 UnpackSize;
  Byte Flags;
  bool KeepSolid;
  int SolidIndex;
```

Static Analysis with CodeQL

- CodeQL is possibly a powerful utility, but it may be difficult to leverage it to its full potential for analyzing our target.
- Potential alternative to using LLVM passes to query target.

Static Analysis with FlawFinder

- Running source through Flawfinder gave 500 hits:

```
ANALYSIS SUMMARY:

Hits = 500
Lines analyzed = 240937 in approximately 6.18 seconds (38963 lines/second)
Physical Source Lines of Code (SLOC) = 183216
Hits@level = [0] 131 [1]  41 [2] 439 [3]   9 [4]  11 [5]   0
Hits@level+ = [0+] 631 [1+] 500 [2+] 459 [3+]  20 [4+]  11 [5+]   0
Hits/KSLOC@level+ = [0+] 3.44402 [1+] 2.72902 [2+] 2.50524 [3+] 0.109161 [4+] 0.0600384 [5+]   0
Minimum risk level = 1
```

Static Analysis with FlawFinder

- Vast majority of hits warn of static buffers
- A couple of hits simply reference #undef printf, sprintf statements at the beginning of code

```
./7zip/UI/FileManager/ProgressDialog2.cpp:746:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:757:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:787:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:804:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:837:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:877:  [2] (buffer) wchar_t:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./7zip/UI/FileManager/ProgressDialog2.cpp:1079:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
```

Static Analysis with FlawFinder

- Interestingly, a few instances of unsafe uses of strcpy are flagged. Though probably harmless, this one in particular could potentially incur a segfault or possibly overflow:

```
static void MethodToProp(int method, int chunksSizeBits, NCOM::CPr
{
  if (method >= 0)
  {
    char temp[32];

    if ((unsigned)method < ARRAY_SIZE(k_Methods))
      strcpy(temp, k_Methods[(unsigned)method]);
    else
      ConvertUInt32ToString((UInt32)(unsigned)method, temp);
```

Static Analysis with FlawFinder

- Seeing as we control Flags (part of the volume/file header), and kMethodMask is hardcoded, we could theoretically craft an input that access past the k_Methods array
- Could potentially overflow temp[] since strcpy only stops at null byte

```
  int method = 0;
  int chunkSizeBits = -1;
  if (r.IsCompressed())
  {
    method = vol->Header.GetMethod();
    chunkSizeBits = vol->Header.ChunkSizeBits;
  }
  MethodToProp(method, chunkSizeBits, prop);
}
```

```
unsigned GetMethod() const
{
  if (!IsCompressed())
    return 0;
  UInt32 mask = (Flags & NHeaderFlags::kMethodMask);
  if (mask == 0) return 0;
  if (mask == NHeaderFlags::kXPRESS) return NMethod::kXPRESS;
  if (mask == NHeaderFlags::kLZX) return NMethod::kLZX;
  if (mask == NHeaderFlags::kLZMS) return NMethod::kLZMS;
  if (mask == NHeaderFlags::kXPRESS2) return NMethod::kXPRESS;
  return mask;
}
```

Static Analysis: Takeaways

- Combining multiple tools is crucial
- Being able to perform Data Flow Analysis is extremely powerful in tandem with simpler syntactic analysis tools/dynamic analysis

Thank you!