Source Code Audit on c-ares
for Open Source Technology Improvement Fund (OSTIF)

Final Report and Management Summary

2023-05-22

X41 D-SEC GmbH
Krefelder Str. 123
D-52070 Aachen
Amtsgericht Aachen: HRB19989

https://x41-dsec.de/
info@x41-dsec.de

Organized by the Open Source Technology Improvement Fund

Revision	*Date*	*Change*	*Author(s)*
1	2023-04-14	Final Report	MSc. H. Moesl, Dipl.-Ing. D. Gstir, R. Weinberger, and M. Vervier
2	2023-05-25	Public Report	MSc. H. Moesl, Dipl.-Ing. D. Gstir, R. Weinberger, and M. Vervier

Contents

Dashboard

Target	
Customer	Open Source Technology Improvement Fund (OSTIF)
Name	c-ares
Type	Source Code
Version	commit (f95ca36cde8cbbf90860d7b5403e3efb69f8b0f7)
Engagement	
Type	Source Code Audit
Consultants	3: MSc. H. Moesl, Dipl.-Ing. D. Gstir, and R. Weinberger
Engagement Effort	15 person-days, 2023-03-13 to 2023-04-12
Total issues found	3

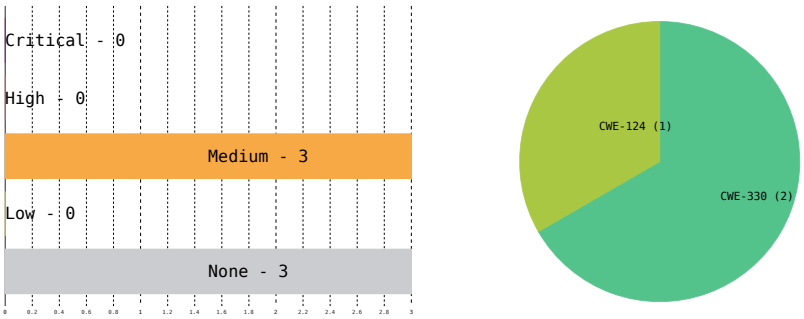

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In March and April 2023, X41 D-Sec GmbH performed a source code audit against c-ares to identify vulnerabilities and weaknesses in the source code and specification. The test was organized by the Open Source Technology Improvement Fund[1].

A total of three vulnerabilities were discovered during the audit by X41. None were rated as having a critical or high severity, three as medium, and none as low. Additionally, three issues without a direct security impact were identified.

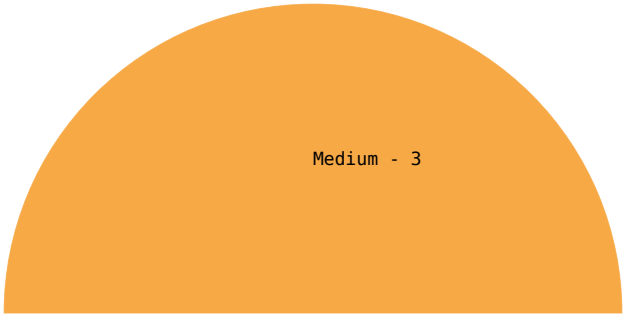

Figure 1.1: Issues and Severity

The *c-ares* library is a library for asynchronous DNS requests (including name resolves) written in C. As it is common for compiled languages, a particular focus has been placed on the identification of typical memory corruption vulnerabilities such as out of bounds memory access, information leaks, or temporal memory safety violations such as use-after-free conditions.

[1] `https://ostif.org`

In a source code audit, the testers receive all available information about the target. The test was performed by three experienced security experts between 2023-03-13 and 2023-04-12.

Despite multiple auditors independently reviewing the same section of the code for better coverage, only three vulnerabilities were identified during the review process. These particular issues pertain to the fact that the c-ares library does not employ a secure random number generator when generating a key that is used for generating DNS query IDs during cross-compilation. While these vulnerabilities do not pose a critical security threat, it could potentially be exploited by attackers to launch DNS cache poisoning attacks on certain targets. Therefore, addressing this vulnerability can add an additional layer of defense and help reduce the potential attack surface of the library. Overall, the current logic for selecting a random source should be redesigned and hardened to use a secure random source on all UNIX and similar systems.

To further improve the security posture, it is encouraged to implement additional security controls, which have been listed as part of the Informational Notes list. These describe potential improvements with regards to missing return value checks.

It is worth emphasizing that the c-ares library was put through rigorous testing by X41's team and, overall, it was found to be highly robust and secure. The testing process revealed that the c-ares library is designed and implemented with great care and attention to detail, and it demonstrated a strong ability to withstand scrutiny. As a result, the overall impression and outcome of this security assessment is very positive.

Again, it must be reiterated that this assessment provided valuable insights into the security posture at the time of testing, but it is important to note that any penetration test is unable to guarantee that the software complex is free of additional bugs.

In terms of dynamic testing, the testing team developed several fuzz testing harnesses. Fuzz testing is, in general, essential for the overall security of the c-ares library project, especially since it is implemented in C, which is often prone to memory corruption vulnerabilities. For the purpose of this test, code coverage driven fuzzing using AFL++ in combination with address space sanitizers (such as ASAN) has been performed. Even though no new issues were identified using the developed harnesses, it is highly recommended to incorporate the developed harnesses into the c-ares library and to make fuzzing a fixed part of the c-ares software project, either using AFL++ or libFuzzer, resulting in better testing coverage.

1.1 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Use of rand() Yields Predictable DNS Query IDs	MEDIUM	CARES-CR-23-01	4.1.1
Use of (Erroneous) RC4 For Random Number Generation	MEDIUM	CARES-CR-23-02	4.1.2
Stack Buffer Underflow in func ares_set_sortlist()	MEDIUM	CARES-CR-23-03	4.1.3
Missing Check of func ares_malloc() Return Value	NONE	CARES-CR-23-100	4.2.1
Missing Check of func snprintf() Return Value	NONE	CARES-CR-23-101	4.2.2
Missing Cleanup in c-ares Tool func adig	NONE	CARES-CR-23-102	4.2.3

Table 1.1: Security-Relevant Findings

2 Introduction

The assessment comprised a security review of the c-ares library, utilizing static source code analysis as well as dynamic testing using dedicated fuzz testing harnesses. The branch in scope for this inspection was the `main` branch with the commit id `f95ca36cde8cbbf90860d7b5403e3efb69f8b0f7`.

At the beginning of the project, an initial kick-off meeting was set up between X41, OSTIF and the maintainers of the library in order to to align on the scope of this engagement. The meeting helped to clarify the expectations of this assessment and also narrowed down the key focus areas.

Throughout the engagement the testers were in communication with OSTIF and the maintainers of the library through a dedicated Signal group. The communication was excellent, and help was provided whenever requested. Generally speaking, OSTIF as well as the maintainers of the library deserve a lot of praise for their overall support and assistance. It was a pleasure for the testing team working with them. From a programming style and software design perspective the code and design is clean and very well written with security in mind.

2.1 Scope

During the kickoff call, the testing team, in collaboration with OSTIF and the maintainers, defined and narrowed down the scope of the testing efforts. It was mutually agreed that the primary focus would be on conducting in-depth static code analysis and incorporating fuzzing support for various functions that had not yet been included in OSS-Fuzz.

In addition to that, the assessment focused a general examination for typical memory corruption vulnerabilities.

2.2 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

A manual approach for code review is used by X41. This process was combined with fuzzing given the nature of c-ares being exposed to parsing of potentially untrustworthy data.

The time allocated to X41 for this code review was sufficient to yield a reasonable coverage of the given scope.

2.2.1 Fuzzing

While conducting a source audit of the c-ares library, it was found that some DNS reply parsers had already been subjected to fuzzing by OSS-Fuzz. Nevertheless, as not all of the pertinent and noteworthy functions had undergone fuzzing via OSS-Fuzz, X41 made the decision to focus on conducting further fuzzing.

For the fuzzing efforts, AFL++[1] was used in two ways / modes:

1. *argv* (command-line) fuzzing of the c-ares tools utilizing AFL++ persistent mode
2. Fuzzing of selected interesting looking functions of c-ares utilizing AFL++ persistent mode

Persistent mode fuzzing is a feature in the AFL++ fuzzer that keeps the target program running in the background and continuously feeds it with new test cases. This is in contrast to the default "one-shot" mode in which the fuzzer launches the target program with each new test case. By utilizing this approach, it was possible to achieve execution speed improvements of 10 to 20 times. Moreover, AFL++ has recently incorporated support for command-line interface (CLI) fuzzing in persistent mode through the *AFL_INIT_ARGV_PERSISTENT* macro, rendering it an ideal choice for the CLI fuzzing of c-ares.

2.2.1.1 Fuzzing Hardware

The fuzzing process was carried out on a system equipped with an AMD Ryzen Threadripper Processor, which boasts 64 cores and 128GB of RAM.

[1] `https://github.com/AFLplusplus/AFLplusplus/`

2.2.1.2 CLI Fuzzing

Considering that c-ares comprises various tools, such as *adig*, *acountry*, and *ahost*, as a component of its code base, X41, decided to conduct command-line interface (CLI) fuzzing against these tools to detect any bugs associated with the parsing of *argv* parameters.

To conduct CLI fuzzing on each of the aforementioned tools, X41 generated a valid set of CLI parameters and utilized them as input test cases for the fuzzer. Additionally, X41 built the code base with address sanitization enabled (*-fsanitize=address*) to detect any memory management errors. To facilitate the fuzzer in quickly finding valid CLI parameters, X41 configured the AFL++ compiler to create a dictionary using the *AFL_LLVM_DICT2FILE* flag based on the compiled C code. Finally, firewall rules were established on the fuzzing machine to redirect DNS requests to `localhost`.

The fuzzer executed each of the aforementioned tools for a total of approximately 5 billion times. Despite the extensive number of executions, X41 was unable to identify any immediate crashes. Given that it can be concluded that the code base for these tools and c-ares is well tested and written with security in mind, at least under the conditions and parameters we used for the fuzzing process.

However, it is worth noting that the absence of immediate crashes does not necessarily imply that the code is free from bugs or vulnerabilities.

2.2.1.3 Fuzzing Selected Functions

During this project, X41 discovered additional noteworthy functions that were not included in the fuzzing efforts of OSS-Fuzz and created dedicated fuzzing harnesses:

- functions doing string manipulation:
  - ***ares__strsplit***
  - ***ares_strdup***
  - ***ares_expand_string***
- functions parsing files or file contents:
  - ***ares__get_hostent***
  - ***ares__read_line***
  - ***ares__readaddrinfo***
- other interesting functions:

- ***config_nameserver***
- ***ares_set_servers_csv***
- ***ares_set_servers_ports_csv***
- ***ares_set_sortlist***

The folders *tests/fuzzinput* and *tests/fuzznames* already contained input test cases provided by OSS-Fuzz, which were utilized as a starting point for generating test cases using the *radamsa* tool[2]. X41 compiled the source code base with address sanitization enabled (*-fsanitize=address*).

The fuzzing harnesses executed each of the aforementioned functions approximately 3 billion times, but no memory memory corruptions or crashes were detected during this process.

Fuzzing the function *are_se_sortlist* resulted in multiple crashes.

2.3 Recommended Further Tests

X41 recommends to subject all newly developed code to regular source code audits. Given the widespread use and criticality of the c-ares library, the code base would profit from recurring security audits as changes within one part of the system may have unintentional security impact to other parts.

[2] `https://gitlab.com/akihe/radamsa`

3 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the test. Business factors and financial risks for Open Source Technology Improvement Fund (OSTIF) are beyond the scope of a penetration test which focuses entirely on technical factors. Yet technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

In total, five different ratings exist, which are as follows:

Severity Rating
None
Low
Medium
High
Critical

A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating ‘none’ are called side findings and are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

3.1 Common Weakness Enumeration

The CWE is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*[1]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

[1] `https://www.mitre.org`

4 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 4.1. Additionally, findings without a direct security impact are documented in Section 4.2.

4.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

4.1.1 CARES-CR-23-01: Use of rand() Yields Predictable DNS Query IDs

Severity:	MEDIUM
CWE:	330 – Use of Insufficiently Random Values
Affected Component:	c-ares/src/lib/ares_init.c:randomize_key(),c-ares/configure.ac

4.1.1.1 Description

X41 found that under certain circumstances, c-ares generates deterministic DNS query IDs.

For generating random DNS query IDs, c-ares uses RC4 as a pseudo random number generator (PRNG). The RC4 key is generated differently depending on the target platform. On ***WIN32*** systems. ***RtlGenRandom()*** is used to get cryptographically secure random numbers. On all other targets, the library tries to use use the file path stored in ***CARES_RANDOM_FILE*** to read random data from. This usually defaults to ***/dev/urandom***, but when c-ares is cross compiled using Autotools (as documented in the source repository's ***INSTALL.md***), this variable is not defined and c-ares falls back to generating the RC4 key using the insecure ***rand()*** function. One case where this happens is when cross compiling c-ares for Android targets with ***aarch64*** architecture.

What is more, c-ares does not seed ***rand()***'s pseudo random number generator (PRNG) using ***srand()***. Consequently, ***rand()*** will always output the same sequence of random numbers which will result in exactly the same sequence of DNS query IDs begin generated. Thus, on targets where ***rand()*** is used an attacker can predict future query IDs and prepare malicious DNS responses ahead of time to launch DNS cache poisoning or similar attacks. Note however that the attacker still needs to be able to know the correct source port of the target host, but this can also be achieved by simply flooding the target with responses from multiple UDP source ports until one packet goes through. A more powerful attacker can launch more elaborate attacks like recording the DNS query and responding faster than the legitimate DNS server.

It is also worth mentioning that when building c-ares using CMake this issue does not occur. However, forks of c-ares that use their own build system might also be affected from a similar issue, since there is not default value for ***CARES_RANDOM_FILE*** and thus ***rand()*** is used as fallback.

Another situation where ***rand()*** is used, is on read errors or short-reads from ***CARES_RANDOM_FILE***.

Should reading from that file fail, or fewer bytes than wanted be read, c-ares fill the remaining bytes of the RC4 key with ***rand()***. This again leads to an RC4 which is more likely to be predictable by a malicious actor.

```
1  static void randomize_key(unsigned char* key,int key_data_len)
2  {
3    int randomized = 0;
4    int counter=0;
5  #ifdef WIN32
6    BOOLEAN res;
7
8    res = RtlGenRandom(key, key_data_len);
9    if (res)
10     randomized = 1;
11
12 #else /* !WIN32 */
13 #  ifdef CARES_RANDOM_FILE
14   FILE *f = fopen(CARES_RANDOM_FILE, "rb");
15   if(f) {
16     setvbuf(f, NULL, _IONBF, 0);
17     counter = aresx_uztosi(fread(key, 1, key_data_len, f));
18     fclose(f);
19   }
20 #  endif
21 #endif /* WIN32 */
22
23   if (!randomized) {
24     for (;counter<key_data_len;counter++)
25       key[counter]=(unsigned char)(rand() % 256);  /* LCOV_EXCL_LINE */
26   }
27 }
```

Listing 4.1: randomize_key Function

```
1  dnl Check for user-specified random device
2  AC_ARG_WITH(random,
3  AS_HELP_STRING([--with-random=FILE],
4                 [read randomness from FILE (default=/dev/urandom)]),
5      [ CARES_RANDOM_FILE="$withval" ],
6      [
7          dnl Check for random device.  If we're cross compiling, we can't
8          dnl check, and it's better to assume it doesn't exist than it is
9          dnl to fail on AC_CHECK_FILE or later.
10         if test "$cross_compiling" = "no"; then
11           AC_CHECK_FILE("/dev/urandom", [ CARES_RANDOM_FILE="/dev/urandom"] )
12         else
13           AC_MSG_WARN([cannot check for /dev/urandom while cross compiling; assuming none])
```

```
14          fi
15
16      ]
17  )
```

Listing 4.2: CARES_RANDOM_FILE Configuration with Autotools

4.1.1.2 Solution Advice

Instead of solely relying on build-time checks when selecting the random source for the generation of the RC4 keys, X41 recommends adding run-time checks which check for the presence of ***/dev/urandom*** and to only fallback to other sources when this file is not present. A superior alternative for ***rand()*** is ***arc4random()*** which is available on BSD derivates and Linux and produces cryptographically secure number on most systems.

X41 also suggest to completely remove the use of RC4 and either replace it with a CSPRNG based on a state-of-the-art cipher like ChaCha20, or use the CSPRNG provided by the OS to retrieve random bytes and use the directly as DNS query ID.

4.1.2 CARES-CR-23-02: Use of (Erroneous) RC4 For Random Number Generation

Severity:	MEDIUM
CWE:	330 – Use of Insufficiently Random Values
Affected Component:	c-ares/src/lib/ares_init.c:init_id_key()

4.1.2.1 Description

X41 found that c-ares uses RC4 in a non-standard way as CSPRNG to generate DNS query IDs. RC4 itself has been shown to be insecure and it's use for CSPRNG is discouraged. What is more, the RC4 implementation in c-ares is erroneous which potentially results in even less security than the original RC4 provides.

Unpredictable DNS query IDs are one mechanism to make DNS cache poisoning attacks harder. Thus, they should be generated in a random, unpredictable way. RC4 has been shown to contain flaws which makes its generated key stream appear non-random (biased). Especially the initial bytes of the key stream have been shown to contain significant biases multiple times [1]. The use of RC4 in protocols like TLS and WPA has been shown to contain practical attack vectors which make RC4 an insecure cipher [2].

Closer analysis of the RC4 key schedule implemented by c-ares in ***init_id_key()*** also showed that it is not implemented correctly: Instead of filling a 31-byte key buffer (***key_data_ptr***) with random bytes, the initial 31 bytes of the RC4 state are overwritten with random data. This is then put through the standard RC4 key schedule. Since ***key_data_ptr*** is filled solely with ***0x00*** bytes, the shuffling of the state is only randomized by the first 31 bytes of the state. Without analyzing this in full detail X41 assumes that the resulting key stream is less random than the original RC4 key schedule.

Another potential source of issues is the fact that the RC4 key stream is initialized once per c-ares channel and never re-keyed. While RC4 key streams have a very long period, it is usually common practice to add new entropy after a certain amount of bytes from the key stream have been used. A practical example for c-ares might be a long-running service that initializes c-ares once using ***ares_init()*** and reuses the resulting ***ares_channel*** for the whole service lifetime.

An attacker might abuse the fact of predictable, or more easily guessable DNS query IDs to run DNS poisoning attacks as described in the previous finding.

[1] `https://link.springer.com/content/pdf/10.1007/3-540-45473-X_13.pdf`
[2] `https://www.rc4nomore.com`

```
int ares_init(ares_channel *channelptr)
{
  return ares_init_options(channelptr, NULL, 0);
}

int ares_init_options(ares_channel *channelptr, struct ares_options *options,
                      int optmask)
{
  // ...

  /* Generate random key */

  if (status == ARES_SUCCESS) {
    status = init_id_key(&channel->id_key, ARES_ID_KEY_LEN);
    if (status == ARES_SUCCESS)
      channel->next_id = ares__generate_new_id(&channel->id_key);
    else
      DEBUGF(fprintf(stderr, "Error: init_id_key failed: %s\n",
                     ares_strerror(status)));
  }

  // ...
}
```

Listing 4.3: *ares_init()* and *ares_ini_options()*

```
static int init_id_key(rc4_key* key,int key_data_len)
{
  unsigned char index1;
  unsigned char index2;
  unsigned char* state;
  short counter;
  unsigned char *key_data_ptr = 0;

  key_data_ptr = ares_malloc(key_data_len);
  if (!key_data_ptr)
    return ARES_ENOMEM;
  memset(key_data_ptr, 0, key_data_len);

  state = &key->state[0];
  for(counter = 0; counter < 256; counter++)
    /* unnecessary AND but it keeps some compilers happier */
    state[counter] = (unsigned char)(counter & 0xff);
  randomize_key(key->state,key_data_len);
  key->x = 0;
  key->y = 0;
  index1 = 0;
```

```
22    index2 = 0;
23    for(counter = 0; counter < 256; counter++)
24    {
25      index2 = (unsigned char)((key_data_ptr[index1] + state[counter] +
26                                index2) % 256);
27      ARES_SWAP_BYTE(&state[counter], &state[index2]);
28
29      index1 = (unsigned char)((index1 + 1) % key_data_len);
30    }
31    ares_free(key_data_ptr);
32    return ARES_SUCCESS;
33  }
```

Listing 4.4: ***init_id_key()*** Function

```
1  unsigned short ares__generate_new_id(rc4_key* key)
2  {
3    unsigned short r=0;
4    rc4(key, (unsigned char *)&r, sizeof(r));
5    return r;
6  }
```

Listing 4.5: ***init_id_key()*** Function

4.1.2.2 Solution Advice

X41 recommends to completely remove the use of RC4 and either replace it with a CSPRNG based on a state-of-the-art cipher like ChaCha20, or use the CSPRNG provided by the OS to retrieve random bytes and use the directly as DNS query ID.

4.1.3 CARES-CR-23-03: Stack Buffer Underflow in *ares_set_sortlist()*

Severity:	MEDIUM
CWE:	124 – Buffer Underwrite ('Buffer Underflow')
Affected Component:	c-ares/src/lib/ares_init.c:ares_set_sortlist()

4.1.3.1 Description

While fuzz testing the c-ares source code, it was found that the function ***ares_set_sortlist()*** suffers from a stack buffer underflow resulting in a crash of the function.

The code snippet depicted in 4.6 shows the fuzzed function. The fuzzer creates all data supplied to the ***sortstr*** parameter.

```
1   int ares_set_sortlist(ares_channel channel, const char *sortstr)
2   {
3     int nsort = 0;
4     struct apattern *sortlist = NULL;
5     int status;
6
7     if (!channel)
8       return ARES_ENODATA;
9
10    status = config_sortlist(&sortlist, &nsort, sortstr);
11    if (status == ARES_SUCCESS && sortlist) {
12      if (channel->sortlist)
13        ares_free(channel->sortlist);
14      channel->sortlist = sortlist;
15      channel->nsort = nsort;
16    }
17    return status;
18  }
```

Listing 4.6: ares_set_sortlist in ares_init.c

When supplying the input data depicted in 4.7 to the function via ***sortstr***, the crash shown in 4.8 happens.

```
1   00000000  30 3a 3a 30 30 3a 30 30  3a 30 30 2f 32           |0::00:00:00/2|
2   0000000d
```

Listing 4.7: Minimized Input Data Crashing the Function

```
    =================================================================
==13490==ERROR: AddressSanitizer: stack-buffer-underflow on address 0x7ffe2e0b7fbf at pc
↪  0x7efcdc57dc3b bp 0x7ffe2e0b7f90 sp 0x7ffe2e0b7f88
WRITE of size 1 at 0x7ffe2e0b7fbf thread T0
    #0 0x7efcdc57dc3a in inet_net_pton_ipv6 /src/src/lib/inet_net_pton.c:360:19
    #1 0x7efcdc57dc3a in ares_inet_net_pton /src/src/lib/inet_net_pton.c:408:13
    #2 0x7efcdc55e031 in config_sortlist /src/src/lib/ares_init.c:1959:19
    #3 0x7efcdc55d885 in ares_set_sortlist /src/src/lib/ares_init.c:2317:12
    #4 0x561a784c4124 in do_fuzzing_test /src/src/lib/fuzzing/harness_set_sortlist.c:50:5
    #5 0x561a784c42a1 in main /src/src/lib/fuzzing/harness_set_sortlist.c:81:9
    #6 0x7efcdc224d8f  (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId:
    ↪  69389d485a9793dbe873f0ea2c93e02efaa9aa3d)
    #7 0x7efcdc224e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29e3f) (BuildId:
    ↪  69389d485a9793dbe873f0ea2c93e02efaa9aa3d)
    #8 0x561a78406414 in _start (/src/fuzz_setsortlist/bin/fuzzer+0x1f414) (BuildId:
    ↪  cd37b97419af59952a0cdac242624047e28bdb38)

Address 0x7ffe2e0b7fbf is located in stack of thread T0 at offset 31 in frame
    #0 0x7efcdc57c41f in ares_inet_net_pton /src/src/lib/inet_net_pton.c:403

  This frame has 1 object(s):
    [32, 48) 'tmp.i' (line 268) <== Memory access at offset 31 underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism,
↪  swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-underflow /src/src/lib/inet_net_pton.c:360:19 in
↪  inet_net_pton_ipv6
Shadow bytes around the buggy address:
  0x100045c0efa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100045c0efb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100045c0efc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100045c0efd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100045c0efe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100045c0eff0: 00 00 00 00 f1 f1 f1[f1]00 00 f3 f3 00 00 00 00
  0x100045c0f000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100045c0f010: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  0x100045c0f020: 00 00 00 00 00 f2 f2 f2 f2 f2 00 00 f2 f2 00 00
  0x100045c0f030: 00 00 f3 f3 f3 f3 f3 f3 00 00 00 00 00 00 00 00
  0x100045c0f040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
```

```
41    Stack right redzone:     f3
42    Stack after return:      f5
43    Stack use after scope:   f8
44    Global redzone:          f9
45    Global init order:       f6
46    Poisoned by user:        f7
47    Container overflow:      fc
48    Array cookie:            ac
49    Intra object redzone:    bb
50    ASan internal:           fe
51    Left alloca redzone:     ca
52    Right alloca redzone:    cb
53 ==13490==ABORTING
```

Listing 4.8: ASAN Output of the Crash

Exploitability of this issue depends on various factors and the context it is used. While it was not reachable from the default tools, the function could be used by software that integrates the libary.

4.1.3.2 Solution Advice

X41 recommends finding and fixing the issue leading to the crash in order to improve the overall stability of the c-ares library.

4.2 Informational Notes

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

4.2.1 CARES-CR-23-100: Missing Check of *ares_malloc()* Return Value

Affected Component:

4.2.1.1 Description

While reviewing the c-ares source code, it was noticed that the function ***ares_get_android_server_list()*** does not properly check the return value of ***ares_malloc()***. Under specific circumstances, ***ares_malloc()*** can fail and return *NULL* to the caller, e.g. when a memory request cannot be satisfied because there is no usable block of memory on the heap of the C runtime or, alternatively, when the C runtime memory management requested more memory from the operating system than available. While this situation is rare, in such a case the application might experience undefined behavior.

```
1  char **ares_get_android_server_list(size_t max_servers,
2                                      size_t *num_servers)
3  {
4      [...]
5      dns_list = ares_malloc(sizeof(*dns_list)*(*num_servers));
6      for (i=0; i<*num_servers; i++)
7      {
8          server = (*env)->CallObjectMethod(env, server_list, android_list_get_mid,
9                                            (jint)i);
10         dns_list[i] = ares_malloc(64);
11         dns_list[i][0] = 0;
12         if (server == NULL)
13         {
14           continue;
15         }
16         str = (*env)->CallObjectMethod(env, server, android_ia_host_addr_mid);
17         ch_server_address = (*env)->GetStringUTFChars(env, str, 0);
18         strncpy(dns_list[i], ch_server_address, 64);
19         (*env)->ReleaseStringUTFChars(env, str, ch_server_address);
20         (*env)->DeleteLocalRef(env, str);
21         (*env)->DeleteLocalRef(env, server);
22     }
```

```
23      [...]
24  }
```

Listing 4.9: ares_get_android_server_list Function

4.2.1.2 Solution Advice

X41 recommends to always check the returned pointer for being ***NULL*** and raise an error when ***ares_malloc()*** fails.

4.2.2 CARES-CR-23-101: Missing Check of *snprintf()* Return Value

Affected Component:

4.2.2.1 Description

While reviewing the c-ares source code, it was observed that two instances do not properly check the return value of ***snprintf()***. ***snprintf()*** can internally use a dynamic buffer and return ***-1*** instead of the string length if ***snprintf()*** runs out of memory. Even though the described behavior only occurs under specific and rare circumstances, in such a case the buffer used for storing the values will potentially be cut off.

```
1   static int init_by_resolv_conf(ares_channel channel)
2   {
3       [...]
4       if (status != ARES_EOF) {
5           char propname[PROP_NAME_MAX];
6           char propvalue[PROP_VALUE_MAX]="";
7           for (i = 1; i <= MAX_DNS_PROPERTIES; i++) {
8             snprintf(propname, sizeof(propname), "%s%u", DNS_PROP_NAME_PREFIX, i);
9             if (__system_property_get(propname, propvalue) < 1) {
10              status = ARES_EOF;
11              break;
12            }
13
14            status = config_nameserver(&servers, &nservers, propvalue);
15            if (status != ARES_SUCCESS)
16              break;
17            status = ARES_EOF;
```

```
18          }
19      }
20      [...]
21  }
```

Listing 4.10: init_by_resolv_conf Function

```
1   static int convert_query (char **name_p, int use_bitstring)
2   {
3       [...]
4       if (ares_inet_pton (AF_INET, *name_p, &addr.addr4) == 1)
5           {
6              unsigned long laddr = ntohl(addr.addr4.s_addr);
7              unsigned long a1 = (laddr >> 24UL) & 0xFFUL;
8              unsigned long a2 = (laddr >> 16UL) & 0xFFUL;
9              unsigned long a3 = (laddr >>  8UL) & 0xFFUL;
10             unsigned long a4 = laddr & 0xFFUL;
11
12             snprintf(new_name, sizeof(new_name), "%lu.%lu.%lu.%lu.in-addr.arpa", a4, a3, a2, a1);
13             *name_p = new_name;
14             return (1);
15      }
16      [...]
17  }
```

Listing 4.11: convert_query Function

4.2.2.2 Solution Advice

It is recommended to properly check the returned value of ***snprintf()*** and ensure that cases when ***snprintf()*** fails are detected.

4.2.3 CARES-CR-23-102: Missing Cleanup in c-ares Tool *adig*

Affected Component: adig.c

4.2.3.1 Description

While reviewing the c-ares source code, it was found that the tool adig does miss cleanup functionality in case of an early abort resulting in memory leakage. While this does not impose an immediate security risk for c-ares, it should be fixed in order to harden the tools provided by c-ares.

The code snippet depicted in 4.12 shows that calls to ***ares_destroy(channel)*** are missing when the application exits prematurely.

```
1   int main(int argc, char **argv)
2   {
3     ares_channel channel;
4     [...]
5
6     status = ares_init_options(&channel, &options, optmask);
7     [...]
8
9     if(servers)
10    {
11      status = ares_set_servers(channel, servers);
12      destroy_addr_list(servers);
13      if (status != ARES_SUCCESS)
14        {
15          fprintf(stderr, "ares_init_options: %s\n",
16                  ares_strerror(status));
17          return 1;
18        }
19    }
20    [...]
21
22    for (;;)
23    {
24      [...]
25      if (count < 0 && (status = SOCKERRNO) != EINVAL)
26      {
27        printf("select fail: %d", status);
28        return 1;
29      }
30    ares_process(channel, &read_fds, &write_fds);
31    }
32
```

```
33        ares_destroy(channel);
34
35        ares_library_cleanup();
36
37        [...]
38      }
```

Listing 4.12: adig.c

4.2.3.2 Solution Advice

X41 recommends to perform proper cleanup of used resources if the applications needs to exit prematurely in order to avoid memory leakage.

5 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Review of the Mozilla Firefox updater[1]
- X41 Browser Security White Paper[2]
- Review of Cryptographic Protocols (Wire)[3]
- Identification of flaws in Fax Machines[4,5]
- Smartcard Stack Fuzzing[6]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[2] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[3] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[4] `https://www.x41-dsec.de/lab/blog/fax/`
[5] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[6] `https://www.x41-dsec.de/lab/blog/smartcards/`

A Fuzzing Harnesses

For additional coverage, this section provides all test harnesses used during the fuzzing campaign of the `c-ares` library.

A.1 CLI Fuzzing of acountry

app:acountry

```
    [...]

    #ifndef HAVE_STRNCASECMP
    #  include "ares_strcasecmp.h"
    #  define strncasecmp(p1,p2,n) ares_strncasecmp(p1,p2,n)
    #endif

    #include <limits.h>
    #include "/AFLplusplus/utils/argv_fuzzing/argv-fuzz-inl.h"

    #ifndef INADDR_NONE
    #define INADDR_NONE 0xffffffff
    #endif

    [...]

    ssize_t fuzz_len;
    unsigned char fuzz_buf[1024000];

    #ifndef __AFL_FUZZ_TESTCASE_LEN
        #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
        #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
        #define __AFL_FUZZ_INIT() void sync(void);
        #define __AFL_LOOP(x) \
        ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
        #define __AFL_INIT() sync()
    #endif
```

```
28
29      __AFL_FUZZ_INIT();
30
31      int main(int argc, char **argv)
32      {
33          ares_channel channel;
34          int    ch, status;
35
36        #if defined(WIN32) && !defined(WATT32)
37          WORD wVersionRequested = MAKEWORD(USE_WINSOCK,USE_WINSOCK);
38          WSADATA wsaData;
39          WSAStartup(wVersionRequested, &wsaData);
40        #endif
41
42        __AFL_INIT();
43
44        unsigned char *fuzzbuf2 = __AFL_FUZZ_TESTCASE_BUF;
45        while (__AFL_LOOP(UINT_MAX))
46        {
47              AFL_INIT_ARGV_PERSISTENT(fuzzbuf2);
48
49              [...] // remaining main function of acountry
50        }
51      }
52
53      [ ... ]
```

Listing A.1: Fuzzing Harness for the Tool acountry

A.2 CLI Fuzzing of adig and ahost

The test harnesses are very similar to the one presented for the tool acountry in A.1.

A.3 Test Harness for Parse Functions

```
1       #include <sys/types.h>
2       #include <unistd.h>
3       #include <limits.h>
4       #include <stdio.h>
5       #include <stdlib.h>
6
7       #include <ares.h>
```

```
8       #include <ares_nameser.h>
9
10      ssize_t fuzz_len;
11      unsigned char fuzz_buf[1024000];
12
13      #ifndef __AFL_FUZZ_TESTCASE_LEN
14
15          #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
16          #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
17          #define __AFL_FUZZ_INIT() void sync(void);
18          #define __AFL_LOOP(x) \
19          ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
20          #define __AFL_INIT() sync()
21
22      #endif
23
24      __AFL_FUZZ_INIT();
25
26      #define MINIMAL_TESTCASE_LEN 4
27
28      #define MAXIMAL_TESTCASE_LEN 4096
29
30      #pragma clang optimize off
31      #pragma GCC optimize("O0")
32
33      //#define FUZZER_GENERATE_TESTCASE 1
34      //#define FUZZER_READ_FROM_FILE 1
35
36      void do_fuzzing_test(const unsigned char *buf, ssize_t len)
37      {
38          unsigned short index = *((unsigned short*)buf);
39          buf += sizeof(unsigned short);
40          len -= sizeof(unsigned short);
41
42          unsigned short count = 12;
43          index = index % count;
44
45          struct hostent *host = NULL;
46
47          switch(index)
48          {
49              case 0:
50              {
51                  struct ares_addrttl info[5];
52                  int count = 5;
53                  ares_parse_a_reply(buf, len, &host, info, &count);
54                  if (host) ares_free_hostent(host);
55                  break;
56              }
57              case 1:
58              {
59                  host = NULL;
```

```
60                  struct ares_addr6ttl info6[5];
61                  int count = 5;
62                  ares_parse_aaaa_reply(buf, len, &host, info6, &count);
63                  if (host) ares_free_hostent(host);
64                  break;
65              }
66              case 2:
67              {
68                  host = NULL;
69                  unsigned char addrv4[4] = {0x10, 0x20, 0x30, 0x40};
70                  ares_parse_ptr_reply(buf, len, addrv4, sizeof(addrv4), AF_INET, &host);
71                  if (host) ares_free_hostent(host);
72                  break;
73              }
74              case 3:
75              {
76                  host = NULL;
77                  ares_parse_ns_reply(buf, len, &host);
78                  if (host) ares_free_hostent(host);
79                  break;
80              }
81              case 4:
82              {
83                  struct ares_srv_reply* srv = NULL;
84                  ares_parse_srv_reply(buf, len, &srv);
85                  if (srv) ares_free_data(srv);
86                  break;
87              }
88              case 5:
89              {
90                  struct ares_mx_reply* mx = NULL;
91                  ares_parse_mx_reply(buf, len, &mx);
92                  if (mx) ares_free_data(mx);
93                  break;
94              }
95              case 6:
96              {
97                  struct ares_txt_reply* txt = NULL;
98                  ares_parse_txt_reply(buf, len, &txt);
99                  if (txt) ares_free_data(txt);
100                 break;
101             }
102             case 7:
103             {
104                 struct ares_soa_reply* soa = NULL;
105                 ares_parse_soa_reply(buf, len, &soa);
106                 if (soa) ares_free_data(soa);
107                 break;
108             }
109             case 8:
110             {
111                 struct ares_naptr_reply* naptr = NULL;
```

```
112                 ares_parse_naptr_reply(buf, len, &naptr);
113                 if (naptr) ares_free_data(naptr);
114                 break;
115             }
116             case 9:
117             {
118                 struct ares_caa_reply* caa = NULL;
119                 ares_parse_caa_reply(buf, len, &caa);
120                 if (caa) ares_free_data(caa);
121                 break;
122             }
123             case 10:
124             {
125                 struct ares_uri_reply* uri = NULL;
126                 ares_parse_uri_reply(buf, len, &uri);
127                 if (uri) ares_free_data(uri);
128                 break;
129             }
130             case 11:
131             {
132                   // Null terminate the data.
133                 char *name = malloc(len + 1);
134                 name[len] = '\0';
135                 memcpy(name, buf, len);
136 
137                 unsigned char *buf2 = NULL;
138                 int buflen2 = 0;
139                 ares_create_query(name, C_IN, T_AAAA, 1234, 0, &buf2, &buflen2, 1024);
140 
141                 free(name);
142                 free(buf2);
143             }
144         }
145     }
146 
147     int main(int argc, char **argv)
148     {
149     #ifdef FUZZER_GENERATE_TESTCASE
150     {
151         unsigned char packet[10] = {0};
152 
153         for(int a = 0; a < 10; a++) {
154             packet[a] = rand() % 256;
155         }
156 
157         FILE *fp = fopen("testcase", "wb");
158         if (fp)
159         {
160             fwrite(packet, 1, sizeof(packet), fp);
161             fclose(fp);
162         }
163         else
```

```
164         {
165             printf("Error opening file\n");
166         }
167         return 0;
168     }
169     #endif
170 
171     #ifdef FUZZER_READ_FROM_FILE
172     {
173         if (argc != 2)
174         {
175             printf("Usage: fuzzer <testcase_from_file>\n");
176             return 0;
177         }
178 
179         FILE *fp = fopen(argv[1], "rb");
180         if (fp)
181         {
182             ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
183             fclose(fp);
184 
185             printf("Read %ld bytes\n", read);
186             do_fuzzing_test(fuzz_buf, read);
187         }
188         else
189         {
190             printf("Error reading from file!\n");
191         }
192 
193         return 0;
194     }
195     #endif
196 
197         ssize_t len;
198         unsigned char *buf;
199 
200         __AFL_INIT();
201         buf = __AFL_FUZZ_TESTCASE_BUF;
202 
203         while (__AFL_LOOP(UINT_MAX))
204         {
205             len = __AFL_FUZZ_TESTCASE_LEN;
206             if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
207                 continue;
208 
209             unsigned char *buf_new = buf;
210 
211             do_fuzzing_test(buf_new, len);
212         }
213 
214         return 0;
215     }
```

Listing A.2: Fuzzing harness for the parse functions

A.4 Test Harness String Operation Functions

```
#include <sys/types.h>
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <ares.h>
#include <ares_nameser.h>

ssize_t fuzz_len;
unsigned char fuzz_buf[1024000];

#ifndef __AFL_FUZZ_TESTCASE_LEN

    #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
    #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
    #define __AFL_FUZZ_INIT() void sync(void);
    #define __AFL_LOOP(x) \
    ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
    #define __AFL_INIT() sync()

#endif

__AFL_FUZZ_INIT();

#define MINIMAL_TESTCASE_LEN 4

#define MAXIMAL_TESTCASE_LEN 4096

#pragma clang optimize off
#pragma GCC optimize("O0")

//#define FUZZER_GENERATE_TESTCASE 1
//#define FUZZER_READ_FROM_FILE 1

extern char **ares__strsplit(const char *in, const char *delms, size_t *num_elm);
extern void ares__strsplit_free(char **elms, size_t num_elm);

extern char *ares_strdup(const char *s1);
```

```
41
42      void do_strsplit_test(const unsigned char *buf, ssize_t len)
43      {
44          size_t count = 0;
45          char **result = ares__strsplit(buf, ", ", &count);
46
47          printf("Count: %ld\n", count);
48
49          ares__strsplit_free(result, count);
50      }
51
52      void do_strdup_test(const unsigned char *buf, ssize_t len)
53      {
54          char *dup = ares_strdup(buf);
55          free(dup);
56      }
57
58      int main(int argc, char **argv)
59      {
60      #ifdef FUZZER_GENERATE_TESTCASE
61      {
62          unsigned char *teststr  = "this,is,a,test";
63
64          FILE *fp = fopen("testcase", "wb");
65          if (fp)
66          {
67              fwrite(teststr, strlen(teststr), 1, fp);
68              unsigned char eof = '\0';
69              fwrite(&eof, 1, 1, fp);
70              fclose(fp);
71          }
72
73          return 0;
74      }
75      #endif
76
77      #ifdef FUZZER_READ_FROM_FILE
78      {
79          if (argc != 2)
80          {
81              printf("Usage: fuzzer <testcase_from_file>\n");
82              return 0;
83          }
84
85          FILE *fp = fopen(argv[1], "rb");
86          if (fp)
87          {
88              ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
89              fclose(fp);
90
91              do_strsplit_test(fuzz_buf, read);
92          }
```

```
93          else
94          {
95              printf("Error reading from file!\n");
96          }
97
98          return 0;
99      }
100     #endif
101
102         ssize_t len;
103         unsigned char *buf;
104
105         __AFL_INIT();
106         buf = __AFL_FUZZ_TESTCASE_BUF;
107
108         while (__AFL_LOOP(UINT_MAX))
109         {
110             len = __AFL_FUZZ_TESTCASE_LEN;
111             if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
112                 continue;
113
114             unsigned char *buf_new = buf;
115
116             do_strsplit_test(buf_new, len);
117
118             buf_new = buf;
119             do_strdup_test(buf_new, len);
120         }
121
122         return 0;
123     }
```

Listing A.3: Fuzzing Harness for String Operation Functions ares__strsplit, ares_strdup

```
1       #include <sys/types.h>
2       #include <unistd.h>
3       #include <limits.h>
4       #include <stdio.h>
5       #include <stdlib.h>
6       #include <string.h>
7
8       #include <ares.h>
9       #include <ares_nameser.h>
10
11      ssize_t fuzz_len;
12      unsigned char fuzz_buf[1024000];
13
14      #ifndef __AFL_FUZZ_TESTCASE_LEN
```

```
15
16          #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
17          #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
18          #define __AFL_FUZZ_INIT() void sync(void);
19          #define __AFL_LOOP(x) \
20          ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
21          #define __AFL_INIT() sync()
22
23      #endif
24
25      __AFL_FUZZ_INIT();
26
27      #define MINIMAL_TESTCASE_LEN 4
28
29      #define MAXIMAL_TESTCASE_LEN 4096
30
31      #pragma clang optimize off
32      #pragma GCC optimize("O0")
33
34      //#define FUZZER_GENERATE_TESTCASE 1
35      //#define FUZZER_READ_FROM_FILE 1
36
37      int ares_expand_string(const unsigned char *encoded,
38                             const unsigned char *abuf,
39                             int alen,
40                             unsigned char **s,
41                             long *enclen);
42
43      void do_fuzzing_test(const unsigned char *buf, ssize_t len)
44      {
45          unsigned char *s = NULL;
46          long length = 0;
47
48          ares_expand_string(buf, buf + 1, len, &s, &length);
49
50          printf("%s\n", s);
51          free(s);
52      }
53
54      int main(int argc, char **argv)
55      {
56      #ifdef FUZZER_GENERATE_TESTCASE
57      {
58          unsigned char *teststr  = "this_is_a_test";
59
60          FILE *fp = fopen("testcase", "wb");
61          if (fp)
62          {
63              char len = (char)strlen(teststr);
64
65              fwrite(&len, 1, 1, fp);
66              fwrite(teststr, len, 1, fp);
```

```
67              fclose(fp);
68          }
69
70          return 0;
71      }
72      #endif
73
74      #ifdef FUZZER_READ_FROM_FILE
75      {
76          FILE *fp = fopen(argv[1], "rb");
77          if (fp)
78          {
79              ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
80              fclose(fp);
81
82              do_fuzzing_test(fuzz_buf, read);
83          }
84
85          return 0;
86      }
87      #endif
88
89          ssize_t len;
90          unsigned char *buf;
91
92          __AFL_INIT();
93          buf = __AFL_FUZZ_TESTCASE_BUF;
94
95          while (__AFL_LOOP(UINT_MAX))
96          {
97              len = __AFL_FUZZ_TESTCASE_LEN;
98              if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
99                  continue;
100
101             unsigned char *buf_new = buf;
102
103             do_fuzzing_test(buf_new, len);
104         }
105
106         return 0;
107     }
```

Listing A.4: Fuzzing Harness for ares_expand_string

A.5 Test Harness for File and String Parsing Functions

```
#include <sys/types.h>
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <ctype.h>

#include <ares.h>
#include <ares_nameser.h>

#define ISSPACE(x)  (isspace((int)  ((unsigned char)x)))
#define ISDIGIT(x)  (isdigit((int)  ((unsigned char)x)))
#define ISALNUM(x)  (isalnum((int)  ((unsigned char)x)))
#define ISXDIGIT(x) (isxdigit((int) ((unsigned char)x)))
#define ISGRAPH(x)  (isgraph((int)  ((unsigned char)x)))
#define ISALPHA(x)  (isalpha((int)  ((unsigned char)x)))
#define ISPRINT(x)  (isprint((int)  ((unsigned char)x)))
#define ISUPPER(x)  (isupper((int)  ((unsigned char)x)))
#define ISLOWER(x)  (islower((int)  ((unsigned char)x)))
#define ISASCII(x)  (isascii((int)  ((unsigned char)x)))

#define ISBLANK(x)  (int)((((unsigned char)x) == ' ') || \
                          (((unsigned char)x) == '\t'))

#define TOLOWER(x)  (tolower((int)  ((unsigned char)x)))

ssize_t fuzz_len;
unsigned char fuzz_buf[1024000];

#ifndef __AFL_FUZZ_TESTCASE_LEN

    #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
    #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
    #define __AFL_FUZZ_INIT() void sync(void);
    #define __AFL_LOOP(x) \
    ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
    #define __AFL_INIT() sync()

#endif

__AFL_FUZZ_INIT();

#define MINIMAL_TESTCASE_LEN 4

```

```
49  #define MAXIMAL_TESTCASE_LEN 4096
50
51  #pragma clang optimize off
52  #pragma GCC optimize("O0")
53
54  extern int ares__readaddrinfo(FILE *fp,
55                                const char *name,
56                                unsigned short port,
57                                const struct ares_addrinfo_hints *hints,
58                                struct ares_addrinfo *ai);
59
60  extern int ares__get_hostent(FILE *fp, int family, struct hostent **host);
61
62  extern int ares__read_line(FILE *fp, char **buf, size_t *bufsize);
63
64  char *try_config(char *s, const char *opt, char scc)
65  {
66    size_t len;
67    char *p;
68    char *q;
69
70    if (!s || !opt)
71      /* no line or no option */
72      return NULL;  /* LCOV_EXCL_LINE */
73
74    /* Hash '#' character is always used as primary comment char, additionally
75       a not-NUL secondary comment char will be considered when specified. */
76
77    /* trim line comment */
78    p = s;
79    if(scc)
80      while (*p && (*p != '#') && (*p != scc))
81        p++;
82    else
83      while (*p && (*p != '#'))
84        p++;
85    *p = '\0';
86
87    /* trim trailing whitespace */
88    q = p - 1;
89    while ((q >= s) && ISSPACE(*q))
90      q--;
91    *++q = '\0';
92
93    /* skip leading whitespace */
94    p = s;
95    while (*p && ISSPACE(*p))
96      p++;
97
98    if (!*p)
99      /* empty line */
100     return NULL;
```

```
101
102         if ((len = strlen(opt)) == 0)
103           /* empty option */
104           return NULL;  /* LCOV_EXCL_LINE */
105
106         if (strncmp(p, opt, len) != 0)
107           /* line and option do not match */
108           return NULL;
109
110         /* skip over given option name */
111         p += len;
112
113         if (!*p)
114           /* no option value */
115           return NULL;  /* LCOV_EXCL_LINE */
116
117         if ((opt[len-1] != ':') && (opt[len-1] != '=') && !ISSPACE(*p))
118           /* whitespace between option name and value is mandatory
119              for given option names which do not end with ':' or '=' */
120           return NULL;
121
122         /* skip over whitespace */
123         while (*p && ISSPACE(*p))
124           p++;
125
126         if (!*p)
127           /* no option value */
128           return NULL;
129
130         /* return pointer to option value */
131         return p;
132       }
133
134
135       void do_fuzzing_test(FILE *fp)
136       {
137           fseek(fp, 0, SEEK_SET);
138
139           struct ares_addrinfo empty_addrinfo = {
140               NULL, /* cnames */
141               NULL, /* nodes */
142               NULL  /* name */
143           };
144
145           const struct ares_addrinfo_hints hints = { ARES_AI_CANONNAME, AF_UNSPEC, 0, 0 };
146           struct ares_addrinfo *ai = malloc(sizeof(struct ares_addrinfo));
147           if (!ai)
148               return;
149
150           *ai = empty_addrinfo;
151
152           ares__readaddrinfo(fp, "test", 0, &hints, ai);
```

```
153         ares_freeaddrinfo(ai);
154     }
155
156     void do_fuzzing_test2(FILE *fp)
157     {
158         fseek(fp, 0, SEEK_SET);
159
160         struct hostent *host = malloc(sizeof(struct hostent));
161
162         ares__get_hostent(fp, AF_INET, &host);
163
164         free(host);
165     }
166
167     void do_fuzzing_test3(FILE *fp)
168     {
169         char *line = NULL;
170         int status;
171         size_t addrlen, linesize, naliases;
172
173         fseek(fp, 0, SEEK_SET);
174
175         while ((status = ares__read_line(fp, &line, &linesize)) == ARES_SUCCESS)
176         {
177             try_config(line, "domain", ';');
178             try_config(line, "hosts:", '\0');
179         }
180     }
181
182     int main(int argc, char **argv)
183     {
184         ssize_t len;
185         unsigned char *buf;
186
187         __AFL_INIT();
188         buf = __AFL_FUZZ_TESTCASE_BUF;
189
190         while (__AFL_LOOP(UINT_MAX))
191         {
192             len = __AFL_FUZZ_TESTCASE_LEN;
193             if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
194                 continue;
195
196             int status = ares_library_init(ARES_LIB_INIT_ALL);
197             if (status != ARES_SUCCESS) {
198                 return 1;
199             }
200
201             FILE *fp = fmemopen(buf, len, "r");
202             if (fp)
203             {
204                 do_fuzzing_test(fp);
```

```
205                 do_fuzzing_test2(fp);
206                 do_fuzzing_test3(fp);
207                 fclose(fp);
208             }
209
210             ares_library_cleanup();
211         }
212
213         return 0;
214     }
```

Listing A.5: Fuzzing Harness for Parsing Files and File Contents

A.6 Other Test Harnesses

```
1   #include <sys/types.h>
2   #include <unistd.h>
3   #include <limits.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7
8   #include <ares.h>
9   #include <ares_nameser.h>
10
11  ssize_t fuzz_len;
12  unsigned char fuzz_buf[1024000];
13
14  #ifndef __AFL_FUZZ_TESTCASE_LEN
15
16      #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
17      #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
18      #define __AFL_FUZZ_INIT() void sync(void);
19      #define __AFL_LOOP(x) \
20      ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
21      #define __AFL_INIT() sync()
22
23  #endif
24
25  __AFL_FUZZ_INIT();
26
27  #define MINIMAL_TESTCASE_LEN 4
28
29  #define MAXIMAL_TESTCASE_LEN 4096
30
31  #pragma clang optimize off
```

```
32      #pragma GCC optimize("O0")
33
34      //#define FUZZER_GENERATE_TESTCASE 1
35      //#define FUZZER_READ_FROM_FILE 1
36
37      int ares_set_servers_csv(ares_channel channel,
38                               const char* _csv);
39      int ares_set_servers_ports_csv(ares_channel channel,
40                                     const char* _csv);
41
42      extern int ares_init(ares_channel *channelptr);
43      extern void ares_destroy(ares_channel channel);
44
45      void do_fuzzing_test(const char *buf, ssize_t len)
46      {
47          ares_channel channel;
48          if (ares_init(&channel) != ARES_SUCCESS)
49          {
50              abort();
51          }
52
53          ares_set_servers_csv(channel, buf);
54
55          ares_destroy(channel);
56      }
57
58      void do_fuzzing_test2(const char *buf, ssize_t len)
59      {
60          ares_channel channel;
61          if (ares_init(&channel) != ARES_SUCCESS)
62          {
63              abort();
64          }
65
66          ares_set_servers_ports_csv(channel, buf);
67
68          ares_destroy(channel);
69      }
70
71      int main(int argc, char **argv)
72      {
73      #ifdef FUZZER_GENERATE_TESTCASE
74      {
75          unsigned char *teststr  = "127.0.0.1:80,267.98.12,[fe80::1%lo0]:53";
76
77          FILE *fp = fopen("testcase", "wb");
78          if (fp)
79          {
80              fwrite(teststr, strlen(teststr) + 1, 1, fp);
81              fclose(fp);
82          }
83
```

```
84          return 0;
85      }
86      #endif
87
88      #ifdef FUZZER_READ_FROM_FILE
89      {
90          FILE *fp = fopen(argv[1], "rb");
91          if (fp)
92          {
93              ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
94              fclose(fp);
95
96
97              do_fuzzing_test(fuzz_buf, read);
98
99              do_fuzzing_test2(fuzz_buf, read);
100         }
101
102         return 0;
103     }
104     #endif
105
106         ssize_t len;
107         unsigned char *buf;
108
109         __AFL_INIT();
110         buf = __AFL_FUZZ_TESTCASE_BUF;
111
112         while (__AFL_LOOP(UINT_MAX))
113         {
114             len = __AFL_FUZZ_TESTCASE_LEN;
115             if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
116                 continue;
117
118             int status = ares_library_init(ARES_LIB_INIT_ALL);
119             if (status != ARES_SUCCESS) {
120                 return 1;
121             }
122
123             unsigned char *buf_new = buf;
124
125             do_fuzzing_test(buf_new, len);
126
127             buf_new = buf;
128
129             do_fuzzing_test2(buf_new, len);
130
131             ares_library_cleanup();
132         }
133
134         return 0;
135     }
```

Listing A.6: Fuzzing Harness for ares_set_servers_csv and ares_set_servers_ports_csv

```
#include <sys/types.h>
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <ares.h>
#include <ares_nameser.h>



ssize_t fuzz_len;
unsigned char fuzz_buf[1024000];

#ifndef __AFL_FUZZ_TESTCASE_LEN

    #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
    #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
    #define __AFL_FUZZ_INIT() void sync(void);
    #define __AFL_LOOP(x) \
    ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
    #define __AFL_INIT() sync()

#endif

__AFL_FUZZ_INIT();

#define MINIMAL_TESTCASE_LEN 4

#define MAXIMAL_TESTCASE_LEN 4096

#pragma clang optimize off
#pragma GCC optimize("O0")


//#define FUZZER_GENERATE_TESTCASE 1
//#define FUZZER_READ_FROM_FILE 1

extern int ares_init(ares_channel *channelptr);
extern void ares_destroy(ares_channel channel);

extern int fuzz_config_nameserver(char *str);

void do_fuzzing_test(const char *buf, ssize_t len)
```

```
46      {
47          // function was added in ares_init.c for fuzzing
48          // calls the static function config_nameserver
49          fuzz_config_nameserver(buf);
50      }
51
52      int main(int argc, char **argv)
53      {
54      #ifdef FUZZER_GENERATE_TESTCASE
55      {
56          unsigned char *teststr  = "11.43.43/24/00192.168.33333";
57
58          FILE *fp = fopen("testcase", "wb");
59          if (fp)
60          {
61              fwrite(teststr, strlen(teststr), 1, fp);
62              fclose(fp);
63          }
64
65          return 0;
66      }
67      #endif
68
69      #ifdef FUZZER_READ_FROM_FILE
70      {
71          FILE *fp = fopen(argv[1], "rb");
72          if (fp)
73          {
74              ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
75              fclose(fp);
76
77              do_fuzzing_test(fuzz_buf, read);
78          }
79
80          return 0;
81      }
82      #endif
83
84          ssize_t len;
85          unsigned char *buf;
86
87          __AFL_INIT();
88          buf = __AFL_FUZZ_TESTCASE_BUF;
89
90          while (__AFL_LOOP(UINT_MAX))
91          {
92              len = __AFL_FUZZ_TESTCASE_LEN;
93              if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
94                  continue;
95
96              int status = ares_library_init(ARES_LIB_INIT_ALL);
97              if (status != ARES_SUCCESS) {
```

```
98                  return 1;
99              }
100
101             unsigned char *buf_new = buf;
102
103             do_fuzzing_test(buf_new, len);
104
105             ares_library_cleanup();
106         }
107
108         return 0;
109     }
```

Listing A.7: Fuzzing Harness for cfg_nameserver

A.7 Crashing Test Harness ares_set_sortlist

```
1   #include <sys/types.h>
2   #include <unistd.h>
3   #include <limits.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7
8   #include <ares.h>
9   #include <ares_nameser.h>
10
11  ssize_t fuzz_len;
12  unsigned char fuzz_buf[1024000];
13
14  #ifndef __AFL_FUZZ_TESTCASE_LEN
15
16      #define __AFL_FUZZ_TESTCASE_LEN fuzz_len
17      #define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
18      #define __AFL_FUZZ_INIT() void sync(void);
19      #define __AFL_LOOP(x) \
20      ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
21      #define __AFL_INIT() sync()
22
23  #endif
24
25  __AFL_FUZZ_INIT();
26
27  #define MINIMAL_TESTCASE_LEN 4
28
29  #define MAXIMAL_TESTCASE_LEN 4096
```

```
30
31      #pragma clang optimize off
32      #pragma GCC optimize("O0")
33
34      //#define FUZZER_GENERATE_TESTCASE 1
35      //#define FUZZER_READ_FROM_FILE 1
36
37      extern int ares_set_sortlist(ares_channel channel, const char *sortstr);
38
39      extern int ares_init(ares_channel *channelptr);
40      extern void ares_destroy(ares_channel channel);
41
42      void do_fuzzing_test(const char *buf, ssize_t len)
43      {
44          ares_channel channel;
45          if (ares_init(&channel) != ARES_SUCCESS)
46          {
47              abort();
48          }
49
50          ares_set_sortlist(channel, buf);
51
52          ares_destroy(channel);
53      }
54
55      int main(int argc, char **argv)
56      {
57      #ifdef FUZZER_GENERATE_TESTCASE
58      {
59          unsigned char *teststr  = "11.43.43/24/00192.168.33333";
60
61          FILE *fp = fopen("testcase", "wb");
62          if (fp)
63          {
64              fwrite(teststr, strlen(teststr) + 1, 1, fp);
65              fclose(fp);
66          }
67
68          return 0;
69      }
70      #endif
71
72      #ifdef FUZZER_READ_FROM_FILE
73      {
74          FILE *fp = fopen(argv[1], "rb");
75          if (fp)
76          {
77              ssize_t read = fread(fuzz_buf, 1, sizeof(fuzz_buf), fp);
78              fclose(fp);
79
80
81              do_fuzzing_test(fuzz_buf, read);
```

```
82          }
83
84          return 0;
85      }
86      #endif
87
88          ssize_t len;
89          unsigned char *buf;
90
91          __AFL_INIT();
92          buf = __AFL_FUZZ_TESTCASE_BUF;
93
94          while (__AFL_LOOP(UINT_MAX))
95          {
96              len = __AFL_FUZZ_TESTCASE_LEN;
97              if (len < MINIMAL_TESTCASE_LEN || len > MAXIMAL_TESTCASE_LEN)
98                  continue;
99
100             int status = ares_library_init(ARES_LIB_INIT_ALL);
101             if (status != ARES_SUCCESS) {
102                 return 1;
103             }
104
105             unsigned char *buf_new = buf;
106
107             do_fuzzing_test(buf_new, len);
108
109             ares_library_cleanup();
110         }
111
112         return 0;
113     }
```

Listing A.8: Fuzzing Harness for ares_set_sortlist