Least Authority
Freedom Matters

Security Audit Report for the Mozilla Secure Open Source Fund

Knot DNS

Overview

Mozilla SOS Fund has requested Least Authority perform a security audit of Knot DNS (www.knot-dns.cz) and the Knot Resolver (www.knot-resolver.cz). Currently, Knot DNS is deployed on K-root and L-root servers and it's being used to serve some top level domains (.cz, .dk, .nl, .fr, .cl), along with other common users of it. The Knot Resolver is a DNS resolver intended to be deployed from small network routers to big resolver farms serving millions of customers. The resolver shares support libraries with the Knot DNS server.

Coverage

Target Code and Revision

For this audit, we reviewed the latest stable releases of Knot DNS and Knot Resolver code found at:

www.knot-dns.cz and www.knot-resolver.cz

Specifically, we examined the Git revisions:

Knot DNS:

`03c53dea4dd14678528c678b57b0af687891a93c`

Knot Resolver:

`16d5d5de480756b5c537a0b9bd695b7d9bd4ee84`

All file references in this document use Unix-style paths relative to the project's root directory.

Dependencies

Although our primary focus was on the application code, we examined dependency code and behavior when it was relevant to a particular line of investigation. In general, we made the assumption that dependencies implemented their APIs securely, i.e. we focused on bugs in the usage of dependencies rather than in the dependencies themselves.

Scope

Our investigation focused on the following areas:

- Mistakes which can't be found by analyzers (Coverity, Clang Static Analyzer, Valgrind, AddressSanitizer, American Fuzzy Lop) such as CVE-2017-11104
- Attacks that are able to crash the service, create indefinite loops in the resolution process or otherwise degrade service, therefore impacting reliability; such as a simple null pointer deref or algorithmic complexity attack against the hash table
- Holes in the DNSSEC validation logic, allowing an attacker to fake e.g. TLSA records in DNS (RFC 6698) or SSHFP
- In particular these security critical areas will be reviewed:
    - (knotd, kresd) Packet parsing and relating untrusted input data processing
    - (knotd) DNSSEC operations, offline signing, and the online signing module
    - (knotd) Access control and TSIG protocol implementation
    - (kresd) DNSSEC validation
- As with any network server written in C, watching out for anything resulting in remote code execution via untrusted inputs
- General DNS library (libknot) and DNSSEC cryptography related library (libdnssec)

Manual Code Review

In manually reviewing the code, we looked for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also kept an eye out for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits.

The files we manually reviewed included:

- For knot-dns, all files under src/ directory, excluding utils/ and zscanner/ which were assumed to only be used with trusted inputs.
- For knot-resolver, all files under contrib/, lib/, daemon/, and modules/ were examined.

Not in Scope

The code is actively tested by a set of unit tests and functional tests, so these will not be suggested unless specific to the areas of concern above.

Convenient utilities (utils) and zone file parser (zscanner) will be excluded from the audit as they are used on the trusted side.

Findings

Code Quality

Overall, we found the code to be well structured and cleanly written. Additionally Knot makes good use of available tools, such as fuzzers and compiler sanitizers.

Issues

We list the issues we found in the code in the order we found them.

Issue A: Non-Cryptographic Hash Used for DNS cookies

Severity: Low
Reference: *knot-dns/src/libknot/cookies*

Impact: Using a non-cryptographic hash like FNV64 for DNS cookies may allow an attacker to recover the server secret, which could result in server impersonation attacks.

Technical Details: FNV64 is used for DNS cookies.

Remediation: Using FNV64 is recommended by RFC 7873, but it would still be preferable to use either HMAC-SHA256, the alternate recommendation in the RFC, or else a cryptographic PRF such as SipHash.

Issue B: Timing Channel in DNS Cookie Comparisons

Severity: Low
Reference: *knot-dns/src/libknot/cookies/client.c* line 58, `knot_cc_check`

Impact: DNS cookie comparisons lead to a small timing channel that might allow an attacker to iteratively guess expected values.

Technical Details: The DNS cookie is compared with memcmp.

Remediation: The DNS cookies should be treated as one would any other authentication code, and compared using a function that does not leak information via side channels (e.g. the existing `const_time_memcmp` function).

Issue C: Weak RSA keys allowed

Severity: Medium
Reference: *knot-dns/src/dnssec/lib/key/algorithm.c*

Impact: Since weak RSA keys can be trivially factored by private entities (see for instance https://eprint.iacr.org/2015/1000.pdf which demonstrates factoring such keys using $75 worth of compute time on Amazon EC2), they should not be allowed for any purpose.

Technical Details: `dnssec_algorithm_key_size_range` allows RSA keys as small as 512 bits.

Remediation: Disallow the use of weak keys by increasing the specified limits.

Issue D: Insufficient Build Hardening

Severity: Low
Reference: *knot-dns/configure.ac*

Impact: In some configurations the daemon may not make use of all the exploit mitigations offered by the compiler and operating system. If a flaw is found, an attacker may have an easier time exploiting the issue than is necessary.

Technical Details: The default build does not enable flags that can help harden the daemon against attacks. Flags like `-fstack-protector` protect against stack overflows, `-D_FORTIFY_SOURCE` on glibc systems enables additional runtime and compile time checks, and -fPIE creates position independent executables which increases the effectiveness of address space randomization.

Remediation: It would be best to utilize flags which can harden the daemon against attacks directly so that so that all users are protected by these additional countermeasures. On some Linux distributions such features are enabled by default.

Issue E: Hash Function Collisions

Severity: Medium
Reference: *knot-dns/src/contrib/hhash.c*

Impact: Since the hash table is shared across all zones, this may allow an attacker who can control some zone inputs to "knock out" a victim zone.

Technical Details: The hhash (hopscotch hash) hash table uses MurmurHash3 hash function. This is vulnerable to trivial collisions, i.e. an attacker can easily create many thousands of inputs which result in the same hash value. The hash table designed used in hhash will return an error if too many colliding keys are inserted. In our tests this occurred after 32 duplicates.

By creating many sets of duplicating keys (e.g., 10000 sets of 32 colliding keys) an attacker may be able to cause serious slowdowns in hash table processing.

The only use of hhash currently is in the zone database. If zone entries can be created such that Murmurhash3 collides,`knot_zonedb_insert` can fail. In addition the return value of knot_zonedb_insert is not checked for an error return, so the hash table collisions will not be noticed.

Remediation: Replace Murmurhash3 with SipHash using a random 128-bit key generated at startup (e.g. created with /dev/urandom or gnutls_rnd)

Issue F: Missing Error Check Causing Crash

Severity: Low
Reference: *knot-dns/src/libknot/rdataset.c*

Impact: A server crash is possible. It is unclear if a remote attacker can cause this condition to arise.

Technical Details: In the function knot_rdataset_reserve, the following sequence of code occurs

```
    rrs->rr_count++;

        // We have to initialise the 'size' field in the reserved
space.
        knot_rdata_t *rr = knot_rdataset_at(rrs, rrs->rr_count -
1);
        assert(rr);
        knot_rdata_set_rdlen(rr, size);
```

If the uint16_t value rr_count is the maximum value, the increment will overflow it to zero. Then knot_rdata_at(rrs, 65535) will return NULL because the pos argument will be greater than the (overflowed) rr_count. This will cause either an assertion failure or (in non-debug builds) a NULL pointer deref in knot_rdata_set_rdlen.

Remediation: Verify at runtime that no integer overflow occurs.

Issue G: Use of assert macro for error checking

Severity: Medium
Reference: *src/knot/nsec-chain.c, src/knot/dnssec/rrset-sign.c*

Impact: While assert is great for some purposes, it doesn't work well for error checking. If enabled, failing asserts can crash the entire process. And if disabled, critical checks may be skipped. If this is true then Knot would be unsafe to compile with NDEBUG (for example a downstream user or distributor might set this flag).

Technical Details: It seems in some cases Knot currently relies on `assert` to verify inputs that are otherwise unchecked. Portions of the code use the `assert` macro heavily.

In *src/knot/nsec-chain.c*:

```
    assert(to->owner);
        size_t next_owner_size = knot_dname_size(to->owner);
        size_t rdata_size = next_owner_size +
dnssec_nsec_bitmap_size(rr_types);
        uint8_t rdata[rdata_size];
        memcpy(rdata, to->owner, next_owner_size);
```

If `asserts` are enabled, this can crash. If asserts are not enabled, and `to->owner` is null, then `knot_dname_size` will return `KNOT_EINVAL`. `KNOT_EINVAL` is `-EINVAL` (from the system errno), for example on Linux this will have value -22. In this particular case the code would presumably just crash due to the null deref of `to->owner`. However it's easy to see how slightly different code flow could result in a stack based buffer overflow.

Another example, in *src/knot/dnssec/rrset-sign.c*

```
#define RRSIG_RDATA_SIGNER_OFFSET 18

size_t rrsig_rdata_header_size(key):
...
        assert(size == RRSIG_RDATA_SIGNER_OFFSET);
        const uint8_t *signer = dnssec_key_get_dname(key);
        assert(signer);
        size += knot_dname_size(signer);
        return size;
```

So if its argument key is NULL, `rrsig_rdata_header_size` returns `RRSIG_RDATA_SIGNER_OFFSET - 22` (or -4 wrapped to `size_t`)

```
    size_t header_size = rrsig_rdata_header_size(key);
        assert(header_size != 0);
 ...
        uint8_t header[header_size];
 ...
       size_t rrsig_size = header_size + signature.size;
        uint8_t rrsig[rrsig_size];
        memcpy(rrsig, header, header_size);
        memcpy(rrsig + header_size, signature.data,
signature.size);
```

On most Unix systems `EINVAL` is 22, so this will crash due to stack exhaustion

when the very large (wrapped around) `size_t` is returned by `rrsig_rdata_header_size`.

Remediation: The developers should carefully audit the uses of `assert`. It seems likely that many of the situations covered by `assert` should instead be changed to return an error code.

Issue H: Not Checking Return Value For Error

Severity: Low
Reference: *knot-resolver/lib/cache.c*

Impact: In the event of memory exhaustion, the resolver may crash.

Technical Details: Twice (in lines 246 and 356) `malloc` is called without checking that that return value is not null.

Remediation: Check return values of any functions which may fail.

Issue I: Weak PRNG

Severity: Low
Reference: *knot-resolver/lib/utils.c*, *knot-resolver/contrib/ccan/isaac*

Impact: ISAAC was designed as a cryptographic RNG, but it has not been well studied and what research has been done on it does not seem promising (see for example https://eprint.iacr.org/2006/438.pdf which demonstrates that ISAAC has many internal states with bad statistical properties).

Technical Details: The resolver uses the ISAAC PRNG for various purposes including generating the DNS query id field, which is critical for preventing blind forgery attacks.

Remediation: It would be better to either use the GnuTLS RNG directly, or else switch to a more modern PRNG such as ChaCha.

Issue J: Integer Overflow

Severity: Low
Reference: *knot-resolver/lib/utils.c*

Impact: The function `kr_strcatdup` has an integer overflow that could result in a heap overflow. This issue is not exploitable in the current codebase because attacker-controlled inputs are not passed to `kr_strcatdup` at any point.

Technical Details: First the `size_t total_len` is computed by calling `strlen` on each of the `varargs`.

```
    for (unsigned i = 0; i < n; ++i) {
                char *item = va_arg(vl, char *);
                total_len += strlen_safe(item);
        }
```

Then an output buffer is allocated, note the second potential integer overflow here

```
    result = malloc(total_len + 1);
```

Finally the outputs are copied out:

```
    for (unsigned i = 0; i < n; ++i) {
                char *item = va_arg(vl, char *);
                if (item) {
                        size_t len = strlen(item);
                        memcpy(stream, item, len + 1);
                        stream += len;
                }
        }
```

For example on a system with 32-bit `size_t`, if `kr_strcatdup` was called with 4 pointers to the same 1 GB string, then all strings would succeed, the `total_len` field would overflow, leading to a malloc of a too-short buffer, followed by a heap overflow.

Remediation: This is easily fixed by changing the loop that computes `total_len`:

```
      for (unsigned i = 0; i < n; ++i) {
                char *item = va_arg(vl, char *);
                size_t with_this_item = total_len +
strlen_safe(item);
                if(with_this_item < total_len)
                     return NULL; // overflow!
                total_len = with_this_item;
        }
```

Issue K: Integer Overflow

Severity: Medium
Reference: *knot-resolver/lib/cache.c*, line 233, `kr_cache_insert`

Impact: There does not seem to be any avenue for a remote attacker to cause this function to be called with such a large value.

Technical Details: This function adds an input (`data.len`, a `size_t`) plus a constant (`sizeof(kr_cache_entry)`)

```
knot_db_val_t entry = { NULL, sizeof(*header) + data.len };

    ...
    auto_free char *buffer = malloc(entry.len);
    entry.data = buffer;
    entry_write(entry.data, header, data);
```

If `data.len` is close to the maximum value of `size_t`, the addition will overflow resulting in a short `malloc` followed by a heap overflow in `entry_write`.

Remediation: Either verify that the addition does not overflow, or alternately simply reject caching values larger than some predefined limit (e.g. 1 megabyte).

Suggestions

Suggestion 1: Redundant Operation

Severity: Informational

Synopsis: In *knot-dns/src/libknot/tsig-op.c*, function `check_digest`, the memset at line 541 seems redundant, since it is copied over immediately by the memcpy:

```
memset(wire_to_sign, 0, size);
memcpy(wire_to_sign, wire, size);
```

Suggestion 2: Google's OSS-Fuzz

Severity: Informational

Synopsis: Consider applying to Google's OSS-Fuzz, which runs fuzzers for critical open source projects on a large number of machines.

Project Team

Jack Lloyd: Lead Reviewer
Jack has over 15 years of experience as a software developer and security auditor. He has worked on projects ranging from VoIP applications to automated trading platforms. As a FIPS-140 reviewer, he examined the security of dozens of proprietary crypto implementations. He is also the author of the Botan cryptography library.

David Stainton: Supporting Reviewer

David is a proponent of the langsec and cypherpunks movements and a code contributor to the Tor and Tahoe-LAFS projects. In the past he wrote a TCP analysis tool to detect injection attacks. Currently he researches and developes mix networks, which are a type of privacy preserving network with a very different threat model than Tor.

Liz Steininger: Point of Contact, Administration
Liz is a supporter of open source software that encourages transparency and access to information, along with software that enables individuals to freely express themselves and retain the ability to control their own information. She has over 15 years of experience as a Program and Project Manager, Strategist and Analyst working towards these goals.

Least Authority Audit Team: Additional Reviewing & Support
The Least Authority team has skills for reviewing code in C, C++, Golang, Python, Haskell, Rust, Node.js and JavaScript for common security vulnerabilities and specific attack vectors. The team has experience reviewing implementations of crypto protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.