Deadwood (MaraDNS 3.5.0036) Applies Full Fixed-Interval Retry Period to Any Unresolvable Nameserver, Exhausting Upstream Connection Slots and Causing Resolver Unavailability

Low **samboy** published **GHSA-cfc6-vhrv-62cj** last month

Package

Deadwood

Affected versions

3.5.0036

Patched versions

None

Description

Title

Deadwood (MaraDNS 3.5.0036) Applies Full Fixed-Interval Retry Period to Any Unresolvable Nameserver, Exhausting Upstream Connection Slots and Causing Resolver Unavailability

CWE

- **CWE-400: Uncontrolled Resource Consumption** (primary) — `dwx_handle_ns_refer_connect()` in `DwRecurse.c` unconditionally sets a per-connection deadline of `now + timeout_seconds` (default: 8 seconds) every time a NS referral is followed, regardless of whether the upstream server has already responded with a deterministic failure (such as NXDOMAIN for the NS hostname). When the NS address is immediately known to be unresolvable, Deadwood still waits the full 8-second interval before retrying. With `num_retries=5` (default), each such resolution attempt occupies one upstream connection slot (`rem[]` entry) for up to 40 seconds. Since the total slot pool is bounded by `maxprocs` (default: 50 in many deployments), 50 concurrent queries to zones with unresolvable NS servers can exhaust all available upstream connection slots, causing subsequent queries from all clients to receive SERVFAIL immediately.

- **CWE-834: Excessive Iteration** (secondary) — `handle_expired()` in `DwSocket.c` retries the upstream connection unconditionally as long as `rem[a].retries > 0`, applying a new fixed-deadline of `now + timeout_seconds` after each attempt. The retry condition does not distinguish between a true network timeout (no response received) and an immediate deterministic failure (response received in under 1ms indicating the NS address cannot exist). As a result, Deadwood iterates through the full `num_retries` cycle even when the resolution outcome is already determined, holding the connection slot for the entire duration.

CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H — Score: 6.5 (Medium)

Metric	Value	Rationale
Attack Vector	Network	The behavior is triggerable remotely by any party who can configure a DNS zone whose NS record is unresolvable and who can induce clients using Deadwood to query that zone
Attack Complexity	Low	No special network positioning is required. The trigger condition (any zone with an unreachable or unresolvable NS server) arises naturally from both deliberate configuration and accidental misconfiguration
Privileges Required	None	Registering a domain with a non-responsive NS address requires no privileges on the Deadwood instance
User Interaction	Required	End-user DNS queries for the affected zone must be routed through Deadwood for the behavior to manifest
Scope	Unchanged	Impact is confined to the Deadwood resolver instance itself
Confidentiality	None	No data leakage observed
Integrity	None	No data modification observed
Availability	High	With 50 concurrent queries for a zone whose NS is unresolvable (matching the default `maxprocs=50`), all upstream connection slots are occupied for up to 40 seconds. During this period, any new client query requiring upstream resolution receives SERVFAIL immediately via Deadwood's overload handler. The resolver is effectively unavailable for non-cached queries for the duration of the slot-occupancy window, which is renewable.

Summary

During security research into DNS resolver behavior under edge-case NS delegation configurations, we identified a behavioral issue in Deadwood 3.5.0036 (the recursive resolver component of MaraDNS 3.5.0036) whereby any zone whose authoritative nameserver address cannot be resolved causes Deadwood to hold an upstream connection slot for up to 40 seconds — even when the resolution failure is known immediately.

The root cause is in `dwx_handle_ns_refer_connect()` in `DwRecurse.c`, which sets the per-connection deadline timer to `now + timeout_seconds` (8 seconds by default) every time a NS referral is followed. This timer is set unconditionally: it does not account for whether the upstream server has already responded quickly (e.g., NXDOMAIN for the NS hostname returned within 1ms). When the NS address is immediately determined to be unresolvable, Deadwood still waits the full 8-second interval before declaring the attempt failed and retrying. With `num_retries=5`, each such query occupies a connection slot for approximately 5 × 8 = 40 seconds.

We confirmed this behavior with two separate DNS configurations:

1. **Cyclic NS delegation** — zone A delegates to `ns.B` and zone B delegates to `ns.A`, with no glue records provided for either NS hostname.
2. **Simple lame NS delegation** — zone C delegates to `ns.nonexistent-outside.`, a hostname that does not exist in any reachable zone.

Both configurations produced identical behavior: responses arrived at Deadwood in under 1 millisecond (Deadwood did reach the authoritative server and received a deterministic failure response), yet Deadwood waited the full 40 seconds before returning a result to the client. The 1:1 timing ratio between these two configurations confirms that the 40-second delay is not attributable to the cyclic structure of case 1 — it is a general property of Deadwood's retry logic that applies whenever a NS address is unresolvable, regardless of the reason.

The upstream connection slot pool (`rem[]`, bounded by `maxprocs`) is Deadwood's only resource for concurrent upstream resolution. With `maxprocs=50`, 50 concurrent queries to an affected zone occupy all available slots for up to 40 seconds. During this period, Deadwood's overload handler (`handle_overload=1`) returns SERVFAIL immediately for any new query requiring upstream resolution. The resolver is effectively unavailable for non-cached queries until the occupied slots are freed.

Importantly, this behavior does not require any deliberate configuration from an operator managing the Deadwood instance: it can be triggered by any external DNS zone whose NS servers happen to be unreachable, including zones affected by registrar misconfiguration, expired NS delegations, or routine infrastructure outages. This means the condition may arise unintentionally in production environments.

Reporter: Qifan Zhang, Sr Staff Researcher, Palo Alto Networks (qzhang@paloaltonetworks.com)

MaraDNS/Deadwood version affected

```
MaraDNS 3.5.0036 / Deadwood 3.5.0036
Built from source (maradns-3.5.0036.tar.xz) on Debian bookworm with:
  cd maradns-3.5.0036/deadwood-3.5.0036/src
  ./make.version.h
  make -f Makefile.sl6
```

```
Running in Docker (debian:12-slim runtime)
Tested: 2026-03-16
```

The behavior is in `deadwood-3.5.0036/src/DwSocket.c` (`handle_expired()`, `kill_expired()`) and `deadwood-3.5.0036/src/DwRecurse.c` (`dwx_handle_ns_refer_connect()`). We expect it to affect other recent Deadwood versions that use the same timer-based retry architecture.

The behavior is present in the **default configuration**. The key parameters are `timeout_seconds` (default: 2 on some builds, 8 in others), `num_retries` (default: 5), and `maxprocs` (typically 50 or higher). Higher values of either `timeout_seconds` or `num_retries` increase slot occupancy duration proportionally.

Steps to reproduce

Setup:

1. Configure a DNS zone with a nameserver that is unresolvable — either because the NS hostname itself does not exist, or because the server at the NS address does not respond on port 53:

   Option A — Lame NS (NS hostname unresolvable):

   ```
   ; Zone delegation with a NS hostname outside any reachable zone
   test.example.  300  IN  NS  ns.unreachable-outside-any-zone.
   ; ns.unreachable-outside-any-zone. has no A record in any zone we serve
   ```

   Option B — Lame NS (NS IP unreachable):

   ```
   ; Zone delegation where the NS server does not respond
   test.example.  300  IN  NS  ns1.test.example.
   ns1.test.example.  300  IN  A  203.0.113.1   ; TEST-NET-3, does not respond
   ```

2. Configure Deadwood to use a controlled root and recursively resolve queries:

   ```
   # dwood3rc
   bind_address = "0.0.0.0"
   chroot_dir = "/var/deadwood"
   recursive_acl = "0.0.0.0/0"
   maxprocs = 50
   num_retries = 5
   timeout_seconds = 8
   filter_rfc1918 = 0
   root_servers = {}
   root_servers["."] = "<authoritative-server-ip>"
   ```

Observation (single-query behavior):

3. Query Deadwood for a name in the affected zone:

```
dig @<deadwood-ip> victim.test.example. A
```

4. Observe that the response takes approximately 40 seconds to arrive (5 retries × 8 seconds), even though the authoritative server responds immediately with a deterministic failure.

Observation (connection-slot exhaustion):

5. Send 50 concurrent queries to Deadwood for names in the affected zone:

```
for i in $(seq 1 50); do
  dig @<deadwood-ip> victim${i}.test.example. A &
done
wait
```

6. While those queries are pending, send a query for an unrelated, non-cached name:

```
dig @<deadwood-ip> unrelated.other-domain.example. A
```

7. Observe that the unrelated query receives SERVFAIL immediately (from Deadwood's overload handler), rather than being resolved. This demonstrates that all 50 upstream connection slots are occupied and Deadwood cannot perform new upstream lookups.

Control (cached or glued zone):

8. Query Deadwood for a zone whose NS address IS resolvable (e.g., one with proper glue records). Observe that resolution completes normally in under 1 second. This confirms the slot exhaustion is specific to the unresolvable-NS condition.

What is the current *bug* behavior?

When Deadwood encounters a NS referral for a zone whose nameserver address cannot be determined, the following sequence occurs:

`DwRecurse.c` — `dwx_handle_ns_refer_connect()` (approx. line 1737):

```
void dwx_handle_ns_refer_connect(int connection_number, dw_str *packet,
                dw_str *query) {
    /* Timer set unconditionally to now + timeout_seconds.
     * This fires regardless of whether the server responded
     * immediately with a failure or never responded at all. */
    rem[connection_number].die = get_time() +
```

```
            ((int64_t)timeout_seconds << 8);
    make_remote_connection(connection_number, ...);
}
```

`DwSocket.c` — `handle_expired()` (approx. line 1150):

```
int handle_expired(int a) {
    if(rem[a].retries > 0) {
        rem[a].retries--;
        /* Retry unconditionally — no check for whether the previous
         * attempt received an immediate deterministic failure response. */
        make_remote_connection(a, ...);
        rem[a].die = get_time() + ((int64_t)timeout_seconds << 8);
        return 1;   /* keep the slot; try again */
    }
    /* Retries exhausted → free slot → SERVFAIL to client */
    server_fail_noreply(a, local_num);
    reset_rem(a);
    return 0;
}
```

`DwSocket.c` — `kill_expired()` (approx. line 1207):

```
void kill_expired() {
    for(a = 0; a < maxprocs; a++) {
        if(rem[a].die > 0 && rem[a].die < get_time()) {
            if(handle_expired(a) == 1) continue;  /* retry */
            reset_rem(a);                          /* free slot */
        }
    }
    /* Called every 50ms in the main event loop */
}
```

Timing trace for a single unresolvable-NS query (with `timeout_seconds=8`, `num_retries=5`):

```
t=0ms    Client query arrives. rem[a].die = now + 8s.
         Query sent to authoritative server.
t<1ms    Server responds with immediate failure (NXDOMAIN or no usable NS address).
         Response processed. rem[a].die still counting toward 8s.
t=8s     kill_expired() → handle_expired() → rem[a].retries-- (5→4)
         rem[a].die = now + 8s. Another query sent to server.
t<1ms    Server responds immediately again. Timer continues.
t=16s    Retry 3. rem[a].die = now + 8s.
t=24s    Retry 4. rem[a].die = now + 8s.
t=32s    Retry 5. rem[a].die = now + 8s.
t=40s    rem[a].retries == 0 → reset_rem(a) → SERVFAIL to client.
```

Data flow summary:

```
Client query arrives
  → dwx_handle_ns_refer_connect(): die = now + 8s (unconditional)
  → make_remote_connection(): query sent to authoritative NS
  → Server responds in <1ms with NXDOMAIN (immediate failure)
  → Response processed; connection stays open; die still counting
  → 8s later: kill_expired() → handle_expired() → retry
  → ... (5 times) ...
  → 40s total: slot freed, SERVFAIL returned to client

Concurrent: 50 such queries → all rem[] slots occupied
  → New client queries requiring upstream: overload handler → SERVFAIL
  → Deadwood effectively unavailable for non-cached queries for 40s
```

The connection slots remain occupied for the full retry duration even though the resolution outcome (NS address unresolvable) is deterministic from the first server response.

What is the expected *correct* behavior?

When the upstream server responds quickly (within a short threshold, e.g., 100ms) with a deterministic failure indicating that the NS hostname does not exist or cannot be resolved, Deadwood should release the connection slot promptly rather than waiting for the full `timeout_seconds` interval to elapse. The retry logic is appropriate for genuine network timeouts (server does not respond), but not for cases where the server has already communicated a definitive negative outcome.

Reasonable resolver behavior calls for early failure when the outcome is already determined: there is no benefit in waiting the full `timeout_seconds` after the upstream server has clearly indicated within milliseconds that the NS address cannot be resolved.

The expected behavior:

1. Query sent to authoritative NS.
2. Server responds with NXDOMAIN (or equivalent immediate failure) in <1ms.
3. Deadwood recognizes this as a deterministic failure, frees the connection slot, and returns SERVFAIL to the client — all within a few milliseconds.
4. No retry attempts, since the failure is not attributable to a network timeout.

Suggested fix

Fix 1 (Primary): Detect immediate failures and skip the retry timer

Record the time at which each upstream query is sent. In `handle_expired()`, check whether the elapsed time since the query was sent is short enough to indicate the server responded immediately (rather than the request timing out due to network issues). If so, skip the retry and release the slot immediately.

```
/* DwSocket.c — add to rem[] struct */
int64_t query_sent_time;    /* ticks when most recent upstream query was dispatched */
```

```
/* dwx_handle_ns_refer_connect() — record send time */
rem[conn].query_sent_time = get_time();     /* NEW */
rem[conn].die = get_time() + ((int64_t)timeout_seconds << 8);
make_remote_connection(conn, ...);

/* handle_expired() — check for immediate failure */
int handle_expired(int a) {
    int64_t elapsed = get_time() - rem[a].query_sent_time;
    int64_t threshold_100ms = (int64_t)100 << 8;   /* 100ms in ticks */

    bool server_responded_immediately = (elapsed < threshold_100ms);

    if (rem[a].retries > 0 && !server_responded_immediately) {
        /* True timeout: server did not respond. Retry is appropriate. */
        rem[a].retries--;
        rem[a].query_sent_time = get_time();
        rem[a].die = get_time() + ((int64_t)timeout_seconds << 8);
        make_remote_connection(a, ...);
        return 1;
    }

    /* Immediate failure (server responded with unusable answer)
     * OR retries exhausted. Give up; release slot immediately. */
    for (local_num = 0; local_num < rem[a].num_locals; local_num++) {
        server_fail_noreply(a, local_num);
    }
    reset_rem(a);
    return 0;
}
```

With this change, a query whose NS address is NXDOMAIN (server responds in <1ms) would release its connection slot within ~100ms rather than after 40 seconds.

Fix 2 (Supplementary): Configuration mitigation

While a code fix is being prepared, reducing `timeout_seconds` and `num_retries` in `dwood3rc` decreases the maximum slot-occupancy duration:

```
timeout_seconds = 2     # was 8 (reduces max occupancy from 40s to ~12s)
num_retries = 2         # was 5 (reduces retry count)
# New max occupancy: 2 retries × 2s + initial 2s = ~6s per query
```

Increasing `maxprocs` raises the concurrency threshold required to exhaust all slots:

```
maxprocs = 200          # was 50 (DoS requires 200 concurrent queries instead of 5
```

These are operational workarounds and do not address the root cause.

Relevant configuration files

```
# dwood3rc — configuration used in testing
bind_address = "0.0.0.0"
chroot_dir = "/var/deadwood"
recursive_acl = "0.0.0.0/0"
maxprocs = 50
handle_overload = 1
num_retries = 5
timeout_seconds = 8
filter_rfc1918 = 0
root_servers = {}
root_servers["."] = "<authoritative-server-ip>"
```

The behavior is present with the **default values** of `timeout_seconds`, `num_retries`, and `maxprocs`. No special flags are required. `handle_overload = 1` is the default in most deployments and causes SERVFAIL to be returned immediately to clients when slots are exhausted (rather than silently dropping packets), which is what allows the overload condition to be observed cleanly.

Relevant logs

Deadwood does not produce per-query timing output in its standard log format. The following measurements are from the test harness querying Deadwood directly:

Single-query timing measurements (three runs):

DNS zone type	Client-observed response time	Slots occupied
Unresolvable NS (lame delegation)	40.00–40.02s	1 for 40s
Cyclic NS delegation	40.01–40.03s	1 for 40s
Properly glued zone (control)	<1ms	0 (resolved from cache/glue)

The identical 40-second response times for both the lame NS and cyclic NS cases confirm that the slot-occupancy duration is not a function of the NS configuration type — it is a fixed property of the retry timer logic.

Connection-slot exhaustion observation:

When 50 concurrent queries are sent to Deadwood for a zone with an unresolvable NS:

- The 50 concurrent queries each consume one `rem[]` slot for 40 seconds
- An immediately subsequent canary query for an unrelated, non-cached domain receives SERVFAIL in <2ms (from the overload handler, not from actual resolution)
- This confirms that all 50 slots are occupied and Deadwood cannot perform new upstream resolution during the occupancy window

Verbose log excerpt (if `verbose_level = 9` is set in dwood3rc):

Deadwood logs the initial query dispatch and the retry attempts (as "did not respond; trying again") even when the server did respond — illustrating the disconnect between the observed network behavior and the timer logic:

```
[Deadwood] Connection for query <name> did not respond; trying again
[Deadwood] Connection for query <name> did not respond; trying again
[Deadwood] Connection for query <name> did not respond; trying again
[Deadwood] Connection for query <name> did not respond; trying again
[Deadwood] Connection for query <name> did not respond; trying again
```

This log message appears five times at 8-second intervals even though the server responded within 1ms on each attempt, which may also indicate that the log message's text does not accurately reflect the actual failure mode.

PoC availability

A Docker-based proof of concept is available upon request. It comprises a controlled authoritative DNS server, Deadwood 3.5.0036 compiled from source, and a Go test harness that measures response latency and demonstrates connection-slot exhaustion under concurrent query load.

```
cd poc/cyclic_ns_delegation_exhaustion
docker compose up --build --abort-on-container-exit --exit-code-from runner
```

The harness confirms:

1. Single-query response time of ~40 seconds for both lame NS and cyclic NS configurations.
2. `handle_overload` SERVFAIL responses for canary queries when 50+ concurrent unresolvable-NS queries are in flight.
3. Normal sub-1ms resolution for a control zone with properly reachable NS.

Severity

Low

CVE ID

No known CVE

Weaknesses

- CWE-400
- CWE-834

Credits

qifan-sailboat Reporter