We Asked Claude to Audit Sagredo's qmail. It found a RCE.

One prompt, 101 minutes, and a working exploit against a widely deployed qmail fork.

CALIF
APR 16, 2026

> *"Find vulnerabilities in latest version of qmail: https://github.com/sagredo-dev/qmail. Focus on vulnerabilities that could result in RCE or system compromise by processing a crafted email."*

That was the entire prompt.

One hour and forty-one minutes later, our in-house harness had spun up a test environment, audited the codebase, found a remote code execution vulnerability, written a working exploit, generated a patch, and produced a full technical report, all without a human touching a terminal.

The Most Secure Software Ever Written

If you've spent any time around mail servers, you know qmail. And if you know qmail, you know Daniel J. Bernstein.

Most people today know DJB as the cryptographer behind a whole ballroom of dancing ciphers and curves: Salsa20, ChaCha20, Tango20 (okay, not Tango), plus Curve25519 and Ed25519, which now sign roughly every SSH session and TLS handshake on the planet. But a decade before any of that, DJB was the guy who decided email security was a solved problem and then solved it.

He wrote qmail in 1995 as a direct rebuke to Sendmail, which at the time was less a mail transfer agent and more a recurring CVE subscription. qmail was deliberately small and paranoid, splitting mail handling across seven mutually-distrustful Unix users so that a bug in one component couldn't touch another. DJB was confident enough in the result to put up a $500 bounty, later raised to $1,000, for anyone who could find a security hole.

For nearly a decade, nobody could. Then in 2005, Georgi Guninski found an integer overflow in `stralloc_readyplus` that could be triggered on 64-bit systems with absurd amounts of RAM. DJB, being DJB, refused to pay, arguing that nobody actually runs qmail on a machine with gigabytes of memory and no resource limits. Qualys eventually proved it exploitable in 2020, and the "nobody" in question turned out to be "most of the Internet."

Disputed payouts aside, qmail became the canonical example of secure software design, the subject of papers and university courses, and "be like qmail" became shorthand for doing security properly. Then DJB stopped maintaining it: the last release, qmail 1.03, shipped in June 1998, and there has never been a 1.04.

The Ship of Theseus Problem

The Internet, inconveniently, did not stop in 1998. A mail server from the Clinton administration doesn't speak STARTTLS, doesn't know about SPF, DKIM, DMARC, SMTP AUTH, or IPv6, and has no idea what to do about the modern spam ecosystem, so the community did what communities do and started patching.

Over 25+ years, qmail accumulated an enormous orbit of third-party patches: netqmail, qmail-tls, vpopmail integration, CHKUSER, SURBL, and dozens more. Eventually people got tired of applying forty patches in the right order, and consolidated distributions emerged. One of the most popular today is Roberto Puzzanghera's (sagredo) qmail, a batteries-included fork that bundles the patches a modern mail admin actually needs.

The problem is that DJB's security guarantee covered DJB's code, and the thousand-dollar bounty was for qmail 1.03. Every patch bolted on since then was written by someone else, reviewed by someone else (or no one), and merged into a codebase whose original safety arguments may no longer hold. The hull is original, but the rigging is not.

What the Machine Found

Our system zeroed in on a feature called `notlshosts_auto` that was added in October 2024. The idea behind it is reasonable: when qmail tries to deliver mail and the remote server's TLS is broken, you don't want to retry TLS forever, so this feature automatically remembers the bad host by

creating a marker file named after it, and future deliveries skip TLS for that host. The implementation lives in `qmail-remote.c`, inside the TLS error handler `tls_quit()`:

```
sprintf(acfcommand, "/bin/touch %s/control/notlshos
        info->pw_dir, partner_fqdn);
fp = popen(acfcommand, "r");
```


It builds a shell command containing the re
`popen()`. The author wrapped the hostname
thinking that neutralizes shell metacharacte
moment the hostname contains a single qu

The obvious objection is that hostnames ca
that's true of *host names* in the RFC 952 se
labels. On the wire, a DNS label is just a len
arbitrary bytes; RFC 1035 lets you put near
recursive resolvers will happily pass it throu
`dn_expand()` to decode an MX record, some special characters get
escaped, but `'`, `` ` ``, `|`, &, <, and > come through untouched. The net result is that `partner_fqdn`, the string being pasted into a shell command, is attacker-controlled via DNS.

The Kill Chain

1. Attacker registers `evil.com`.
2. Attacker sets its MX record to point at a "hostname" like:

   ```
   x'`id>/tmp/pwned`'y.evil.com
   ```

   That's 29 bytes in the first label, well within the 63-byte limit and perfectly legal on the wire.
3. Attacker points an A record for that name at a server they run, which speaks just enough SMTP to advertise STARTTLS and then deliberately botch the handshake.
4. Victim's qmail server tries to deliver *any* email to `evil.com`: a direct send, a forward, a mailing list bounce, an autoreply, whatever.
5. TLS fails, `tls_quit()` fires, and `popen()` runs:

```
/bin/touch /var/qmail/control/notlshosts/'x'`id>/tmp/pwned`'y.evil.com'
```

6. The shell sees the single quotes close and reopen around a backtick substitution, and dutifully executes `id>/tmp/pwned` as the `qmailr` user.

We've published the full chain (Dockerized repro environment, DNS hook, fake SMTP server, exploit script, patch, and the AI-generated technical report) at github.com/califio/publications/tree/main/MADBugs/qmail.

We reported the issue to Roberto Puzzanghera, who fixed it promptly in commit 749f607 and shipped the fix in v2026.04.07. If you run sagredo's qmail with `notlshosts_auto` enabled, you should upgrade.

The Takeaway

To be clear, this is not a DJB bug. You won't find `popen()` anywhere in qmail 1.03; it lives entirely in a community patch. And as shell injections go, it's not a particularly subtle one. A careful human reviewer would have caught it too.

What's notable is the cost. The input was one sentence and a URL, and the output was a verified exploit, a patch, and a report, with the reasoning in between (that DNS labels carry arbitrary bytes, that `dn_expand()` doesn't escape backticks, that the data flows into `popen()`) worked out unattended. That kind of end-to-end audit used to be expensive enough that most patch collections like this one simply never got reviewed. It isn't expensive anymore, for defenders or for attackers.

The practical conclusion is that this capability is worth pointing at your own code: the stuff you ship, and the dependencies you pull in. If 101 minutes of machine time can find bugs like this, you'd rather they be your 101 minutes than someone else's.

The software that survives the next decade will be the software that was audited by the same thing that's attacking it.

Full technical report, PoC, and patch: github.com/califio/publications/tree/main/MADBugs/qmail.

By Khanh · Launched 3 years ago

1 Like

Discussion about this post