New remote denial of service vulnerability in axTLS zero size x509 certificate parsing

(CVE-2019-9689)

by Robert Hörr (Security Evaluator of the Telekom Security Evaluation Facility)

In addition to CVE-2019-8981 (CVS version 3 score 9.8) a new remote denial of service (DoS) vulnerability (CVE-2019-9689) in the axTLS library for embedded devices (version 2.1.5, http://axtls.sourceforge.net) was discovered on 2019 February 25 with modern fuzzing methods. Unfortunately the vulnerability is not yet fixed.

How security-critical is the vulnerability?

This vulnerability may be used to perform a remote denial of service attack. The goal of a denial of service attack is to disturb the availability of a remote service or server, e.g. any access axTLS library. Hence, this vulnerability is security-critical, if any security relevant process depends on remote availability.

Where is the vulnerability in the source code?

The vulnerability is in the function *process_certificate,* which executes in the following way:

tls1_svr.c:

```
ret = process_certificate(ssl, &ssl->x509_ctx);
```

In this function, the variable *num_certs* can be zero and hence the following memory allocations with *calloc(…),* which are marked in bold, are also zero:

tls1.c:

```
int process_certificate(SSL *ssl, X509_CTX **x509_ctx)
    X509_CTX **certs = 0;
    int *cert_used = 0;

    certs = (X509_CTX**) calloc(num_certs, sizeof(void*));
    cert_used = (int*) calloc(num_certs, sizeof(int));
```

The following write and free operations use the variables *certs* and *cert_used* later, which are marked in bold. The function *ssl_free(...)* is executed to clean up a finished TLS connection.

tls1.c:

```
int process_certificate(SSL *ssl, X509_CTX **x509_ctx)
    *x509_ctx = certs[0];
    cert_used[0] = 1;

EXP_FUNC void STDCALL ssl_free(SSL *ssl)
    x509_free(ssl->x509_ctx);
```

x509.c:

```
void x509_free(X509_CTX *x509_ctx)
    free(x509_ctx->ca_cert_dn[i]);
    free(x509_ctx->cert_dn[i]);
    …
```

Therefore, these operations could e.g. rewrite or remove memory of another process.

How can the vulnerability be exploited?
A TLS certificate handshake message with zero certificates is sent to the target system. In this way, the value of the variable *num_certs* will be zero.

What do we learn from this?
This vulnerability is critical because the received total certification length or the variable *num_certs* is not checked for a minimum size. That is a typical mistake, if “security by design” is not used properly. Especially an implementation of a security protocol like TLS must take into account “security by design”.
Modern fuzzing methods or a source code review must be performed to find vulnerabilities. These can exist mainly because “security by design” is not used. As software is becoming more complex, modern fuzzing appears to be a better approach.