New critical remote denial of service vulnerability in axTLS x509 certificate parsing

(CVE-2019-10013)

by Robert Hörr (e-mail: robert.hoerr@t-sytems.com)
(Security Evaluator of the Telekom Security Evaluation Facility)

In addition to the last remote denial of service vulnerability (CVE-2019-9689) a new critical remote denial of service vulnerability (CVE-2019-10013) in the axTLS library for embedded devices (version 2.1.5, http://axtls.sourceforge.net) was discovered on 2019 March 12 with modern fuzzing methods. Unfortunately the vulnerability is not yet fixed.

How security-critical is the vulnerability?

The vulnerability may be used by a remote attacker to allocate significant amount of memory and processor time. So the availability of a service may be disturbed. This kind of attack is called denial of service (DoS). Therefore, 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 that the result of the function `uint32_t get_asn1_length(…)` is not checked for a minimum or maximum size. Later this result is used to allocate memory. The following lines of code are an example. The interested lines are marked in bold.
Notice that the vulnerability occurs on several places in the source code.

ans1.c:

```
static int asn1_get_printable_str(const uint8_t *buf, int *offset, char **str)
    int len = X509_NOT_OK;
    len = get_asn1_length(buf, offset);

    if (asn1_type == ASN1_UNICODE_STR)
    {
        int i;
        *str = (char *)malloc(len/2+1);     /* allow for null */

        for (i = 0; i < len; i += 2)
            (*str)[i/2] = buf[*offset + i + 1];

        (*str)[len/2] = 0;                  /* null terminate */
    }
    else
    {
        *str = (char *)malloc(len+1);       /* allow for null */
        memcpy(*str, &buf[*offset], len);
        (*str)[len] = 0;                    /* null terminate */
    }
```

The variable **`len`** is not checked and the value of the variable could be 2.147.483.647 (integer maximum value). Hence, the **`malloc`** function would allocate 2.147.483.647 / 2 or 2.147.483.647 byte. After this some data with the same size is copied into this new allocated memory. This leads to a high workload of the system that disturbs other running processes.

What do we learn from this?

As we have already seen through the other vulnerabilities “security by design” is not used properly. Hence, a source code review or modern fuzzing methods must be performed to discover some vulnerabilities. As software is becoming more complex, modern fuzzing appears to be a better approach.