Garmin Forerunner 235: Unbounded string copy in NEWS TVM instruction

Vendors

- Garmin

Affected Products

Forerunner 235 firmware version 7.90

Summary

The ConnectIQ program interpreter trusts the string length provided in the data section of the PRG file. It allocates memory for the string immediate and then copies the string into the TVM object using a function similar to `strcpy`. This copy can exceed the length of the allocated string data and overwrite heap data. A successful exploit would allow a ConnectIQ app store application to escape and perform activities outside the restricted application execution environment.

Mitigation

This issue was fixed by Forerunner 235 software version 8.20.

Credit

This issue was found by Dion Blazakis of Atredis Partners.

References

- https://github.com/atredispartners/advisories/ATREDIS-2020-0006.txt
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27486

Report Timeline

- 2020-04-17: Atredis Partners sent an initial notification to Garmin, including a draft advisory.
- 2020-08-17: Atredis Partners shared a copy of the draft advisory with CERT/CC.
- 2020-10-05: Atredis Partners published this advisory.

Technical Details

The TVM interpreter is responsible for running the application (or `.PRG`) downloaded from the Garmin ConnectIQ app store. The `.PRG` file packages both resources (e.g., images and text) and TVM bytecode needed to run the program. Applications are programmed in the proprietary MonkeyC language and are built into `.PRG` programs via the free Garmin ConnectIQ SDK. Once on the device, the virtual machine ensures the applications are strictly constrained to prevent excess use of memory or computation time. Additionally, the runtime API exposed to each program is constrained based on the type of application installed (e.g., a watchface, a widget, a full application).

The TVM provides the ability to load immediate strings into a runtime object by using the `NEWS` instruction. Upon execution, this instruction creates a `tvm_value` of type `STRING`. The value of the string is loaded from an address provided as a 32-bit operand. The data at the provided address is expected to contain a string definition of the form:

```
uint8_t one; // 0x01
uint16_t length;
uint8_t utf8_string[length];
```

The string data buffer is allocated to hold `length` bytes but then a function similar to `strcpy` is used to populate it. This function will stop when a `NUL` byte is encountered possibly overflowing the buffer beyond the size of the initial allocation.

```
int __fastcall tvm_op_news(struct tvm *ctx)
{
  int tvm_addr_for_string; // r0
```

```
  struct stack_value *v3; // r2
  int result; // r0

  tvm_addr_for_string = tvm_fetch_int((int *)&ctx->pc_ptr);
  v3 = ctx->stack_ptr;
  ctx->stack_ptr = v3 + 1;
  v3[1].type = NULL;
  ctx->stack_ptr->value = 0;
  result = tvm_value_load_string(ctx, tvm_addr_for_string, (int)ctx->stack_ptr);
  if ( !result )
    result = tvm_value_incref(ctx, (struct tvm_value *)ctx->stack_ptr);
  return result;
}

int __fastcall tvm_value_load_string(struct tvm *ctx, int string_def_addr, int string_value_out)
{
  int rv; // r0
  unsigned __int8 *string_def; // [sp+4h] [bp-14h]

  rv = tvm_tvmaddr_to_ptr(ctx, string_def_addr, &string_def);
  if ( !rv )
    rv = tvm_string_def_to_value(ctx, string_def, (unsigned __int8 *)string_value_out, 1);
  return rv;
}

int __fastcall tvm_string_def_to_value(_BYTE *a1, unsigned __int8 *a2, unsigned __int8 *a3, int a4)
{
  _BYTE *v4; // r6
  unsigned __int8 *v5; // r4
  struct tvm_value *v6; // r5
  int result; // r0
  _BYTE *v8; // r4
  int v9; // r6
  __int16 v10; // r0
  int v11; // r3
  int v12; // [sp+4h] [bp-14h]

  v4 = a1;
  v5 = a2;
  v6 = (struct tvm_value *)a3;
  if ( a4 )
  {
    if ( *a2 != 1 )
      return 5;
    v5 = a2 + 1;
  }
  result = tvm_value_string_alloc_by_size((struct tvm *)a1, v5[1] | (*v5 << 8), (int)a3);
  if ( !result )
  {
    result = tvm_value_string_to_ptr(v4, v6, &v12);
    if ( !result )
    {
      v8 = v5 + 2;
      v9 = v12;
      v10 = strlen_utf8(v8);
      v11 = v12;
      *(_WORD *)(v9 + 6) = v10;
      strcpy_(v11 + 8, (unsigned int)v8);
      if ( v6->type == STRING )
        return sub_10DE28(v6);
      return 5;
    }
  }
  return result;
}
```

The `tvm_string_def_to_value` function allocates the string using the size found in memory and then proceeds to `strcpy_` the provided data into the freshly allocated buffer.

Triggering this requires direct bytecode manipulation of a PRG file (or construction of one from scratch). There are a number of additional constraints to turn this into a reliable read/write anything anywhere primitive but they do not seem insurmountable.