Garmin Forerunner 235: Missing bounds check in LGETV and LPUTV TVM instructions

Vendors

- Garmin

Affected Products

Forerunner 235 firmware version 7.90

Summary

The ConnectIQ program interpreter fails to check the index provided when accessing the local variable in the `LGETV` and `LPUTV` instructions. This provides the ability to both read and write memory outside the bounds of the TVM context allocation. It can be leveraged to construct a use-after-free scenario leading to a constrained read/write primitive across the entire MAX32630 address space. 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-2020-0005.txt
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27485

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 is a stack-based virtual machine. A set of instructions, `LGETV` and `LPUTV`, are used to read and write to local variable. The virtual machine maintains a frame pointer used to point at the start of the frame on the stack. The entry of a method will reserve some space on the stack to store local variables. The `LGETV` and `LPUTV` instructions expect a single byte operand specifying the local variable index for that instruction. The implementation does not check that this index is within the allocated local variable space as seen below.

```
int __fastcall tvm_op_lgetv(struct tvm *ctx)
{
  char *pc_at_entry; // r3
  struct stack_value *sp_at_entry; // r1
  int local_var_idx; // t1
  struct stack_value *local_var_ptr; // r2
  struct stack_value *v6; // r5

  pc_at_entry = ctx->pc_ptr;
  sp_at_entry = ctx->stack_ptr;
```

```
  local_var_idx = (unsigned __int8)*pc_at_entry;
  ctx->pc_ptr = pc_at_entry + 1;
  local_var_ptr = &ctx->frame_ptr[local_var_idx + 1];
  ctx->stack_ptr = sp_at_entry + 1;
  sp_at_entry[1] = *local_var_ptr;
  v6 = (struct stack_value *)&ctx->m_0x007b;
  tvm_value_incref(ctx, (struct tvm_value *)ctx->stack_ptr);
  tvm_value_decref(ctx, v6);
  ctx->m_0x007b = (struct tvm_value)*ctx->stack_ptr;
  tvm_value_incref(ctx, (struct tvm_value *)v6);
  return 0;
}
```

The unchecked offset from the `frame_ptr` of the execution context provides a path to both memory access past the end of the TVM context allocation (the stack is allocated at the end of this) and a primitive to construct a use-after-free taking advantage of the way values outside of the valid stack are treated.

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.