Garmin Forerunner 235: Unchecked read outside of TVM stack in DUP

Vendors

- Garmin

Affected Products

Forerunner 235 firmware version 7.90

Summary

The ConnectIQ program interpreter trusts the offset provided for the stack value duplication instruction, `DUP`. The offset is unchecked and memory prior to the start of the execution stack can be read and treated as a TVM object. A succesful exploit could use the vulnerability to leak runtime information such as the heap handle or pointer for a number of TVM context variables. This could help create a reliable exploit when paired with some other vulnerability classes.

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-2020-0007.txt
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27483

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 and maintains a runtime stack of TVM values. The `DUP` instruction allows the running program to duplicate a value from any slot on the stack and push the copy on the top of the stack.

```
int __fastcall tvm_op_dup(struct tvm *ctx)
{
  char *pc; // r1
  struct stack_value *sp; // r2
  int stack_offset; // t1
  struct tvm *ctx:v4; // r3
  int v5; // r0
  struct stack_value v7; // [sp+0h] [bp-10h]

  pc = ctx->pc_ptr;
  sp = ctx->stack_ptr;
  stack_offset = (unsigned __int8)*pc;
  ctx->pc_ptr = pc + 1;
  ctx:v4 = ctx;
  v7 = sp[-stack_offset];
```

```
    v5 = *(_DWORD *)&v7.type;
    ctx:v4->stack_ptr = sp + 1;
    *(_DWORD *)&sp[1].type = v5;
    HIBYTE(sp[1].value) = HIBYTE(v7.value);
    tvm_value_incref(ctx:v4, (struct tvm_value *)&v7);
    return 0;
  }
```

The implementation is not complex. It reads the next byte from the instruction stream, uses this byte as the negative offset to read from the top of the stack, and then copies that value to the next stack entry. Finally, the function increases the reference count in the tvm value.

Triggering this requires direct bytecode manipulation of a PRG file (or construction of one from scratch). The vulnerability is likely only useful when used in conjunction with another vulnerability to provide a reliable read/write primitive.