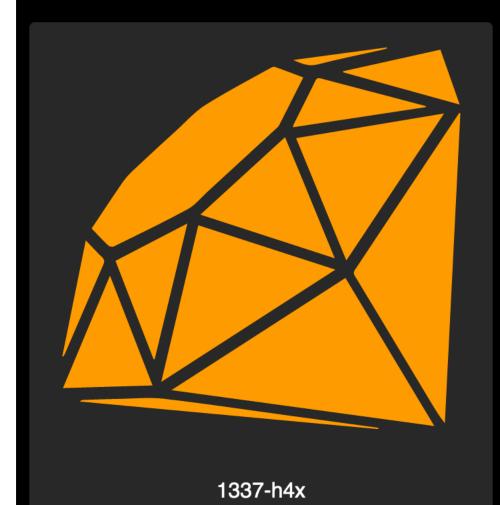
dc2021f-barb-metal

pwnable a/d task





idk...

System Information

DEF CON 2021

Challenge barb-metal Barb Metal



Up To Date v0.0.1

Pwning Maybe

Getting Hacked **Almost Certainly**

Smart Speaker

Active

Online

Thermostat

69 F

Night Temperature

78 F

Day Temperature

Alarm

Online DISARMED

Online

Armed

Console

excute

SHA256: 8c01af87f5103cf43a86bb898f7510f93fd1d6af3ec2e3c261b2d62d20241e0b

Launching payload...

welcome to barbOS...

barb0S> not armed

barb0S> THERM read day friday day 78

barb0S> THERM read night friday niaht 69

barb-metal attack/defend challenge

- this task has a few exploitable vulnerabilites
- in the ctf, teams are expected to patch them
 - payload.bin is patchable (5000 byte patch limit)
- find and exploit one of them!
- patch your bytecode to avoid exploitability. (can't touch the native code).

running the task

kind of hard-ish, run it in docker, it's the easiest

- git clone https://github.com/o-o-overflow/dc2021f-barb-metal-public.git
- cd dc2021f-barb-metal-public/service
- docker build -t barb-metal.
- docker run -p 7829:7829 -d —name barb-metal barb-metal
- docker exec -it barb-metal /bin/bash (get a shell in the container)
- wsprox.py spins up a websockets proxy for communicating with it over regular tcp (port 1338 will talk to ./wsprox.py <host> <port>).

- service (x86 elf) runs "bare metal" in qemu (or service.dbg if you want less of a reversing experience)
- payload.bin mruby/c compiled bytecode

mruby/c

embedded runtime for microcontrollers

- mruby 2.1.0 is bytecodecompatible, you can use mruby 2.1.0's disassembler here (mruby -v -b ./payload.bin) (after chopping off the first 260 bytes of signature and size
- open source, pretty easy to understand how the code works.

There are (at least) 4 exploitable bugs

we will walk through 2, cover the other two just talking through(time permitting)

- I only wrote exploits for 2 of the bugs, but i know teams exploited the other two during the ctf.

- We will go through 1 bug in the native "driver", and 1 bug in the mrubyc runtime (0day at the time).

thermostat write oob write lead to flag disclosure

 chain oob write in thermostat write to disclosure of flag through alarm

 hint (only one instance of alarm is ever created), this is bare metal, aslr is not a thing

```
void __cdecl c_alarm_info(mrbc_vm *vm, mrbc_value *v, int argc)
{
    mrbc_value nnn; // [esp+4h] [ebp-24h]
    mrbc_value ret; // [esp+10h] [ebp-18h] BYREF
    alarm_ptr_t *apt; // [esp+1Ch] [ebp-Ch]

    apt = (alarm_ptr_t *)(v->_anon_0.i + 16);
    mrbc_string_new_cstr(&ret, vm, apt->alrm->ptr);
    nnn = ret;
    mrbc_decref_1(v);
    *v = nnn;
}
```

demo

heap buffer overflow alarm test

- no c code written by me here :) all mrubyc
- arbitrary code execution, all address is rwx
 - (where to store shellcode though? smartspeaker has this feature)
- single mrubyc vm bytecode calls vulnerable function in the runtime
- hint: this is running in x86 32 bit, also mrubyc vm is open source

```
elsif cmd[1] == "settest"
    c2 = split line, " "
    s = c2[-1]
    len = s.size
    i = 0
    while i < len
        @alarm_test[i] = s[i]
        i = i + 1
    end
elsif cmd[1] == "test"
    count = cmd[-1].to_i
    x = @alarm_test * count
    puts x</pre>
```

```
266 OP_GETIV R10 @alarm_test
269 OP_MOVE R11 R8 ; R8:count
272 OP_MUL R10
```

demo

additional bugs

i didn't exploit these, sorry:(

• oob read in thermostat, just read flag byte-by-byte

 c_smartspeaker_queue_popular heap buffer overflow, overwrite pointer of song name with flag buf