```yaml
url: "https://seimei.do/zh/blog/windows-on-arm-rcs380-driver/"
title: "让只有 x64 驱动的设备跑在 Windows on ARM 上——RC-S380 PC/SC 驱动自制记"
description: "Windows on ARM 能仿真运行 x64 应用，但驱动不在仿真范围内。SONY PaSoRi RC-S380 的官方驱
  ↪ 动只有 x64 版，于是我用 UMDF2 虚拟读卡器 + Python 中继的两层结构，自己写了一个 ARM64 原生的 PC/SC
  ↪ 驱动。本文记录这个公开资料极少的领域里的实战经验：绕过 WDK 缺失 MSBuild 集成的问题、未见于文档的必需属
  ↪ 性 SCARD_ATTR_CHANNEL_ID、断开 USB 后缓存伪装成功的故障等。源码以 MIT 协议公开。"
pubDate: 2026-08-10
tags: ["windows", "arm64", "driver", "nfc"]
series: ""
related: ["vmware-fusion-windows-on-arm-devenv"]
graphLabel: "RC-S380 WoA 驱动"
graphWeight: 1
lang: zh
draft: false
ogImage: "/blog/windows-on-arm-rcs380-driver/og-zh.jpg"
```

https://seimei.do/zh/blog/windows-on-arm-rcs380-driver/

我开始自己申请专利，不请代理人。电子申请只能在 Windows 上进行，所以第一步是装好 VMware。

申请似乎需要一台 IC 读卡器，全新购买并不便宜。

日本雅虎拍卖上这种读卡器大量出品、价格便宜，于是我以一口价 895 日元拍下了一台 SONY PaSoRi RC-S380 非接触式读卡器。没有说明书、没有包装盒，带数据线。

问题随后出现。我 Mac 上的 Windows 是 Windows on ARM，官方驱动不支持，读卡器无法使用。我不想为这点事再买硬件，于是自己写了驱动。动手前 Claude Code 估计要四周，实际不到一天就完成了。

就是这个：

bakemocho/rcs380-windows-arm64-driver — 让 SONY PaSoRi RC-S380 在 Windows 11 ARM64 上作为 PC/SC 读卡器工作的自制驱动。UMDF2 虚拟读卡器 + Python 中继结构。

AI 好像总是低估自己的能力。也许是因为它从人类身上学来的？

因为要花钱，所以没有做安装包。请让 AI 帮你安装。

ARM64 版 Windows 不在支持范围内——这一点是买了之后才发现的。当时有三个选择：加钱买现行后继机型 RC-S300（拍卖行情约 3,500 日元）；去网吧用 Windows 电脑办完；或者自己写驱动。我选了最有意思的第三个。

这一趟折腾让我学会了 Windows 驱动的开发方法。这些经验已经派上用场——NT 技研工业委托的数据记录仪维修项目正好能用上。看起来最傻的选择，结果是最好的选择。

顺便推理一下，S380 为什么在拍卖上这么便宜。

- 已经停产，新品价格也崩了（价格.com 上全新最低约 1,200 日元）
- 作为经典机型出货量大，二手市场存量多

図 1　収到的 SONY PaSoRi RC-S380 机身（黑色、带 NFC 标识）与随附的 USB 数据线。没有包装盒和说明书，只有机身和数据线

- 官方软件支持的环境很窄，环境一变就被出手
- 现行的 RC-S300 摆在那里，现在没多少人选旧机型

一句话：会用的人能用，价格却按废品算。

结果是省下了买 S300 的钱。开发驱动烧掉的 AI token，虽然包含在订阅里，成本可能和一台 S300 差不多。即便如此，这些经验和积累可以直接用于下一个驱动，而且过程很快乐。说不定还让满地便宜的 S380 增值了一点。

专利也已经不通过代理人、自力申请了 1 件，另有 4 件正在准备。内容暂时保密，敬请期待。

对技术感兴趣的读者请看这里

Windows on ARM 可以通过仿真运行 x86/x64 应用，但内核态驱动和 **UMDF** 驱动不在仿真范围内，必须是 ARM64 原生才能加载。外设不一定像应用那样直接可用，能不能用取决于厂商是否发布了 ARM64 版驱动。

没有 ARM64 驱动的设备，会一直挂在设备管理器的「其他设备」下，显示代码 28（未安装驱动）。Windows Update 的自动搜索也找不到匹配的驱动。厂商不出，等是等不来的。

我手上变成这种状态的，就是非接触式 IC 读卡器 SONY PaSoRi RC-S380。它的 Windows 官方驱动只有 x64 版。于是我自己写了一个 ARM64 原生的 PC/SC 驱动。先说结果。

- 以虚拟读卡器的身份注册进 Windows 智能卡栈，`SCardTransmit` 可以和卡片完成 APDU 往返（响应 9000）
- 系统重启、USB 断开，都能在无人工干预下自动恢复
- 全部源码以 MIT 协议公开：rcs380-windows-arm64-driver

这篇文章是整个过程的记录。ARM64 上的 UMDF2 驱动开发公开资料很少，关键的坑几乎搜不到，所以连同解法一起写下来。

「USB 能通信」和「PC/SC 能用」之间有一道坎

先破除一个误解。在 Linux 的 pcsc-lite 里，经由 libusb 的用户态驱动可以直接注册为 PC/SC 读卡器。Windows 上同样的做法行不通。

实际上，在开发最初的阶段（Stage 0），我用引用内置 WinUSB 的自制 INF 加自签名测试证书，把 RC-S380 绑到 WinUSB 上，从 nfcpy 直接驱动它。

USB 透传、Port-100 协议、ISO 14443 Type B、ISO-DEP 的 APDU 往返全部打通。到这一步，和卡片的通信本身已经成立。

即便如此，在需要 PC/SC 的应用看来，读卡器依然不存在。Windows 的 PC/SC（SCardSvr / 资源管理器）只认 `Class=SmartCardReader` 的内核态或 UMDF 驱动。

用户态里 USB 通了，只是「能和卡说上话」而已。通往智能卡栈的门还没有打开。

那么 UMDF 能不能写智能卡读卡器？ 能。Microsoft 官方文档明确写了第三方 UMDF 读卡器驱动的 INF 要求（Installing Smart Card Reader Drivers）。要求只有两条。

- `Class=SmartCardReader`
- `UmdfKernelModeClientPolicy=AllowKernelModeClients`

内核态库 smclib 并非必需。Microsoft 自家的内置 CCID 驱动本身就是 UMDF2 写的。

「智能卡读卡器 = 必须内核态」这个成见，动手之前查一手资料就能破除。

分层结构：驱动只做虚拟读卡器

采用的是两段式结构。

```
Windows smart card stack (SCardSvr)
  → 自制 UMDF2 驱动 (ARM64 原生)   ← Windows 自动启动
  → TCP 127.0.0.1:35963            ← 驱动侧 listen
  → Python 中继                     ← 手动启动
  → nfcpy
```

```
→ RC-S380
→ 卡片
```

驱动只扮演虚拟读卡器，实际的射频处理交给用户态的 Python。理由有三。

1. RC-S380 的 Port-100 协议 nfcpy 已经完整实现，在驱动里重写毫无价值
2. nfcpy 是 EUPL-1.1，驱动这边想用 MIT。拆成隔着 TCP 的两个程序，许可证就不会混进同一作品里
3. 方便调试。中继是 Python，不用编译驱动就能对着假的对端做单元测试

层间协议直接采用了 vsmartcard 项目的 vpcd wire protocol。它只是在 TCP 上加 2 字节大端长度前缀的朴素协议，搭上现成规范，中继侧不用做任何协议调研就能写。

不过这个协议有两处并不显然。其一，发起方在层间是反转的：TCP 连接由中继侧 connect（驱动作为服务器 listen/accept），而连接建立后的每次交换由驱动侧先发后收。其二，应答是不对称的。

控制码	含义	应答
`0x00`	powerOff	不得应答
`0x01`	powerOn	不得应答
`0x02`	reset	不得应答
`0x04`	getATR	必须应答
（其他）	APDU	必须应答

在「不得应答」的一侧回了包，字节流就会错位一条消息。麻烦的是症状的呈现方式：当场不坏，只会在之后的某个时刻表现为「双方突然对不上话」。有没有先掌握这张表，排障成本天差地别。

ARM64 构建实务：WDK 的 MSBuild 集成进不了 Build Tools

WDK 10.0.26100.1 起正式支持 ARM64 原生开发（Building ARM64 Drivers）。但真正搭建构建环境时，第一堵墙就在眼前。

WDK 的 Visual Studio 集成以 VSIX 形式提供，而 **VSIX** 只能装进完整版 **Visual Studio**，进不了 Visual Studio Build Tools。也就是说，就算备好 `.vcxproj`，Build Tools 环境下的 `msbuild` 也编不了驱动。

绕行的出路在 UMDF2 自身的性质：UMDF2 驱动本质上是用户态 DLL。放弃 MSBuild 集成，写一个直接调用 `cl.exe` / `link.exe` 的构建脚本就能通。

公开仓库里的 `driver/build-direct.ps1` 就是它。换成 KMDF，这一招是行不通的。

结果反而是好事：摆脱了对完整版 Visual Studio 的依赖，构建的可复现性更高了。

再往后还有一串小墙。

现象	对策
`vs_buildtools` 的 `--log` 参数报错 exit 87	去掉该参数重跑
`WdfDriverStubUm.lib` 链接时要求 `DbgPrintEx`	链接 `ntdll.lib`
`initguid.h` 与 `winioctl.h` 的 GUID 定义冲突（C2374）	本地定义 GUID 绕开
WDF 头文件在 `/W4` 下报 C4324	抑制该警告
字符编码相关的编译错误	加 `/utf-8`

单个都不大，但连踩五个，人就开始怀疑这条路本身是不是走错了。其实全是工具链的已知性质，逐个排掉就能通。

另外要注意，即使在 ARM64 机器上构建，工具链选错也会产出 x64 二进制。为排除这种情况，我确认了两处。

- 编译器：`bin\Hostarm64\arm64\cl.exe` 确实存在
- 产物：`dumpbin` 显示 `AA64 machine (ARM64)`

环境为 Windows 11 24H2 (ARM64) / MSVC 14.44 / Windows SDK 10.0.26100.0 / WDK 10.0.26100.1，整套工具链约 12GB。

为什么不提供安装包
签名的现状也写清楚。正式的分发签名（Partner Center 的证明签名）没有申请，目前用自签名测试证书 + 开启 testsigning 运行。用 `New-FileCatalog` 生成目录并签名，`pnputil` 注册驱动包，`devcon` 创建根枚举设备。

在这种构成下，别人要用这个驱动，就得关闭 Secure Boot 并开启 testsigning。所以公开的是源码而不是安装包——会编译的人用得上，但不是人人都能立刻装上的东西。

踩过的地雷

把开发中踩过、且可以一般化的问题列出来。共同点是：症状指向的位置都不是真正的病因。

地雷 1：不实现 `SCARD_ATTR_CHANNEL_ID`，读卡器注册本身会被拒绝
最重要的一个。现象是读卡器以正确的名字出现在 `SCardListReaders` 里，`SCardConnect` 却全部报 `SCARD_E_UNKNOWN_READER` 失败。

「列表里有，却是 unknown reader」这种矛盾的症状，会引导你去怀疑名字拼写、大小写不一致，但名字查到天亮也解决不了。真正的病因只有一点：没有实现 `SCARD_ATTR_CHANNEL_ID`（`0x00020110`）这个属性。

机制是这样的：PC/SC 资源管理器在添加读卡器时会查询这个属性。返回 `STATUS_NOT_SUPPORTED`，它就记一条 System 事件 610，并拒绝注册这个读卡器。

但名字仍会留在 reader database 里，SCardListReaders 照样列出来。注册失败了，枚举却成功。正是这种错位，把症状伪装成「名字的问题」。

应该返回的值如下。

- 属性定义：SCARD_ATTR_VALUE(SCARD_CLASS_COMMUNICATIONS = 2, 0x0110)
- 值的编码：0xDDDDCCCC（高字 = 连接方式，低字 = 通道号）
- NFC 的连接方式是 0x0100，通道 0 时返回 0x01000000

这个属性明明是注册的必要条件，文档里却没有写明「必需」。而且事件日志只以原始字节的形式记录这次查询，光盯着事件日志永远发现不了拒绝的存在。

定位的关键是驱动自带的文件日志（driver/log.c）。资源管理器按什么顺序问了什么、我们答了什么——把这些用自己的日志排出来，才第一次看到拒绝发生的瞬间。

同一个属性在 vsmartcard 的 Windows 驱动里也没有实现，于是以#324 报告了上游。

有意思的是经过：维护者本人此前已经正确定位到了这个属性，却判断「应该不需要实现」。定位是对的，实现却是必要的。文档里不写「必需」的必需属性，就会造出这种擦肩而过。

地雷 2：诊断工具自己的 bug 制造了同样的症状

验证用的 PowerShell 脚本里混入了一个语言特性导致的 bug：管道结果只有一个元素时会退化成标量字符串。于是 $readers[0] 返回的不是读卡器名，而是名字的第一个字符，工具报告「读卡器 'r' 不存在」。

也就是说，有一段时间，地雷 1 的真因和工具的 bug 在制造同一个症状。我以为在追驱动的 bug，实际有一半时间在追自己的诊断工具。没有「怀疑诊断工具本身」这道工序的诊断，早晚会这样卡死。

地雷 3：缓存伪装出来的成功

v0.3 里发现的现象：物理拔掉 USB，PC/SC 层仍持续返回 PRESENT 和旧的 ATR，只有 SCardConnect 失败。

最初的记录写的是「驱动的 recv 没有超时，挂起并持有锁」。听上去合理，但那是未经验证的假说，事后证明是错的。连同更正一起写在这里。

真因在中继侧。它把感应过一次的卡片缓存起来，之后完全不碰硬件，一直用缓存回答 ATR 请求。USB 断了缓存还在，atr() 一直成功，驱动便一直判定「有卡」。驱动没有挂起，只是持续收到了错误的回答。

修复放进了两层，各司其职。

中继侧每次轮询都用 tag.is_present 确认实体设备存活，USB I/O 出错时关闭前端、每 2 秒重试打开。这是快速检测和恢复的主力。

驱动侧给接收加了 30 秒超时。它是「绝不允许永久挂起」的最后一道防线，实测确认正常运行时从不触发。

超时值的选择是设计判断。压到 5 秒左右，就会制造一种新故障：把合法但耗时的 APDU 误切断。既然快速检测由中继负责，驱动侧就可以放长。决定了故障检测放在哪一层，各层的数值自然就定了。

地雷 4・5：oemNN 编号与依赖语言环境的输出
写脚本和操作手册时容易踩的两个。

pnputil 注册的驱动包会得到 oem9.inf、oem10.inf……这样的编号，而这个编号每次更新都会变。写死进脚本或手册，下次更新就会悄悄指向别的包。

确认应该用 DriverVer，oemNN 当不了标识符。另外旧包不必删除，PnP 会正确选择最新版（实测确认）。

还有一个：pnputil 的输出依赖显示语言。按日语环境的输出字符串写的解析脚本，在英语环境下会悄无声息地坏掉。同理，公开文档里也不要直接贴事件日志的日文文案，改用英文原始字符串。

地雷 6・7：报给上游的两件事，和让 bug 无处容身的结构
做 UMDF2 适配时，在 vsmartcard 的代码里发现并报告了两个问题。

其一是完成状态的类型混用（#335）。UMDF1 用 HRESULT 完成请求，UMDF2 用 NTSTATUS。既有代码把两者混在一起用（比如在 HRESULT 语境里用 STATUS_NO_MEDIA），UMDF2 化需要把所有完成点归一。工作本身是机械的，但漏一处 SCardSvr 就会出错。

其二是取消路径上的迭代器失效（#334）：对 vector 做完 erase 之后还在继续用那个迭代器。

我的驱动干脆不保留后者的结构：不再用自管理的 vector 存放等待插拔的请求，改为交给 UMDF2 的 manual queue（driver/queue.c）。手工的 MarkCancelable 管理消失了，迭代器失效这种 bug 连生存空间都没有了。

不是修 bug，而是造一个 bug 无法存在的结构。这是只有重写时才有的选项。

验证：不经过改动路径的测试，什么也证明不了

新版本要进入「日常实际使用的配置」，条件定为三项测试全部通过。

#	测试	看什么
1	端到端读卡	SCardTransmit 返回 response: 9000
2	系统重启后自动恢复	驱动自动启动 / 生效版本没有回退 / testsigning 保持 / USB 分配保持
3	USB 断开后恢复	断开期间的状态更新，以及零人工干预的自动恢复

变成三项的经过更值得写。最初打算只用「再确认 9000 能返回」当晋级条件。但某个版本的改动全在断开/重开路径上，正常流程的 9000 测试根本不经过那条路径。

那个版本的验证里，Test 1 通过之后，直到 Test 3 才第一次验到这次发布真正要验的行为。如果只靠 Test 1 就晋级，等于抱着「修过的路径可能没修好」的隐患上了日常配置。

一般化地说：把正常流程的连通确认当晋级条件，改动的内容可以一次都没被测过就晋级。不经过改动路径的验证，不是验证。

Test 2 还有一个副产物。重启演练中观察到，中继还没启动时 TCP 端口就已经处于 LISTENING。也就是说驱动在开机时自动加载，连 vpcd 传输层的 TCP 服务器都自己立起来了。

于是确定：重启后需要的手动操作只有启动中继这一条命令，操作手册里删掉了启动驱动的步骤。本来是去确认「能不能恢复」，结果弄清了「能自动恢复到哪一步」。验证同时也是对规格的发现。

v0.3 → v0.4 的差异由 Test 3 确认。

	v0.3	v0.4
断开期间的 PC/SC 状态	停留在过期的 `PRESENT`	立即更新为 `EMPTY | CHANGED`
检测	不检测	1 个轮询周期
恢复	需要手动重启中继	重新连接即可全自动回到 `9000`
日志	无尽的 traceback	一行 `reader not available at usb:054c:06c1 ([Errno 19] No such device)`

刷屏的 traceback 会在下次出事时把真因埋掉。日志的质量也可以列入验证项目。

最后是验证工具侧的两个坑。

- 验证脚本没有返回退出码，失败也可能被调用方当成成功，这是事后才发现的
- `pnputil` 输出的语言依赖解析（地雷 5）差点让「验证过的版本」和「实际生效的版本」错位

「测试通过了」这句话，前提是测试真的跑了。验证工具值得用验证对象同等的严格度对待。

结语

公开仓库 rcs380-windows-arm64-driver 包含 UMDF2 驱动（`driver/`）、Python 中继（`relay/`）和验证工具（`tools/`）的全套。本文各节与仓库文件大致对应如下。

本文内容	仓库对应位置
分层结构与 vpcd 协议	`driver/vpcd.c`, `relay/vpcd_protocol.py`
构建的墙与绕行	`driver/build-direct.ps1`
`SCARD_ATTR_CHANNEL_ID`	`driver/reader.c`, `driver/log.c`
缓存伪装的成功	`relay/card_backend.py`
用 manual queue 做结构性回避	`driver/queue.c`

本文内容	仓库对应位置
三项晋级测试	`tools/pcsc-e2e.ps1`

在 Windows on ARM 上停在代码 28 的设备，只要属于 UMDF2 能覆盖的类型，就存在不等厂商的路。要点有三。

- 「WinUSB 能通信」和「PC/SC 能用」是两回事
- WDK 的 MSBuild 集成不可用时，直接调用 `cl.exe` 也能绕过去
- 不实现 `SCARD_ATTR_CHANNEL_ID`，读卡器注册本身会被拒绝

希望能给尝试同类移植的人做个参考。