Vehicle Diagnostics Adapter
Cybersecurity Concerns with Wireless Connectivity
Edward Larson, Wyatt Ford, Sam Lerner, Dr. Jeremy Daily

This material is based in part upon work supported by the DARPA under Contract Numbers N66001-20-C-4032 and N66001-20-C-4021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA. This paper is released under Distribution Statement “A” (Approved for Public Release, Distribution Unlimited).
Problem Statement

Immediate Problem

• My Vehicle Diagnostic Adapter (VDA) has WiFi and Bluetooth

• WiFi and Bluetooth are unneeded/undesirable in some environments
  • For example, maintaining military vehicles

• Vendor does not (or cannot) remove underpinning software/hardware
  • “Disabled” features can be re-enabled by attackers
  • Attack surface still exists

Broader Problem

• Component X has Features A, B, C

• Features A, B, C are unneeded/undesirable

• Vendor does not (or cannot) remove software/hardware underpinning Feature A, B, C
Research Goals

Immediate Goal
Apply binary patch to VDA firmware that:
  • Removes unwanted feature software: Bluetooth, WiFi
  • Firmware with removed features has no unwanted side effects

Broader Goal
Create automated process to audit and harden Component A
  • Map Features X, Y, Z to binary code and data in firmware images
  • Remove Features X, Y, Z from Component A with binary patch
What is a Vehicle Diagnostics Adapter (VDA)?

System Design

VDA Component Diagram
Open Firmware Reverse Analysis Konsole (OFRAK)

Modular framework to unpack, analyze, modify, and pack binaries

- Component Interface
- Identifier
- Unpacker
- Analyzer
- Modifier
- Packer
- Find/create free space in binaries
- Python APIs
- Graphical User Interface (GUI)
Autotomic Binary Reduction (ABR)

Automated removal of unnecessary or unwanted binary code and data

1. Identify **entry points**
2. Identify **code and data** exclusively control-flow dependent on **entry points**
3. Remove **code and data** from firmware
4. Replace with:
   a. Return patch
   b. Reclaimed space
ABR Workflow

Hardware Teardown

Obtain firmware

Reverse engineer firmware & update process

Identify Features Entry Points

Identify code & data to remove

Remove unwanted features

Repack modified firmware & update

Human Input (for now…)

Automated
Hardware Teardown

Our photo of the VDA after we opened the case, highlighting key components
Obtain Firmware, Reverse Engineer Firmware Update Process

Obtain Firmware
1. Extract firmware update file from Diagnostics tool
2. Use UART to discover memory mapping

Reverse Engineer Update using packet capture (PCAP)
1. Send “\x03” + the size of the firmware update file
2. Send the firmware update file
Reverse Engineer Firmware - RTOS

• No symbols, but …
  • Some software components could be identified by strings
  • Found leaked source online for these components, adds a lot of symbols back!

```c
kernel_data = mqx_init->kmem_start + 0xfU & 0xffffffff;
kernelp_data = kernel_data;
*kernel_data = &INT_401c9eec;
*kernel_data = &PTR_s_Freescale/Freescale_MQX_401c9f08;
bzero(kernel_data, 0x414);

kernel_data = (KERNEL_DATA_STRUCT_PTR) _ALIGN_ADDR_TO_HIGHER_MEM(mqx_init->START_OF_KERNEL_MEMORY);

/* Set the global pointer to the kernel data structure */
_SET_KERNEL_DATA(kernel_data);

/* The following assignments are done to force the linker to include
  • the symbols, which are required by TAD.
  • Note that we should use address of the variable so it is not optimized
  • as direct constant assignment when optimization level is high.
  • Note that counter will be immediately reset to zero on the subsequent
  • _mem_zero call. */
*(volatile void **) kernel_data = (void *) &_mqx_version_number;
*(volatile void **) kernel_data = (void *) &_mqx_vendor;

/* Initialize the kernel data to zero. */
_mem_zero((void *) kernel_data, (_mem_size) sizeof(KERNEL_DATA_STRUCT));
```
Identify Feature Entry Points

5 Generalized Entry Points to WiFi/Bluetooth subsystem

- OS tasks
- UART debug commands
- SDIO driver initialization code
- Marvell SD8787 driver code
- TI Bluetopia library code

```c
if ((wifi_nvmdata->usbblk2_type & BLUETOOTH) == BLUETOOTH) {
    bluetooth_init();
    copy_bt_names_into_nvm();
}
```

```c
bt_Display(s_I/O_Capabilities:_%s, MITH: _%s._013e624,
        (SPTR_s_Display Only_41003964)[DAT_416a8f84], pcVar5);

s_wifi_init 40134397
Op WifiInit
s_Load_SD8787_Wifi_Driver-_Ipconfig_D_4013434a1

const char * driver_version[] =
  "SD8787-%s-M2614" MLAN_RELEASE_VERSION "-GPL" "-" "FP" FNUM ""
```

```c
#include <stdio.h>

void init_wifi(void)
{
    int tries;

    ok = 1;
    if (global_sdio_driver == 0x0) {
        global_sdio_driver = init_sdio_interface();
        tries = 0;
        while ((global_sdio_driver == 0x0 && (tries != 0x0))) {
            wait(1000);
            global_sdio_driver = init_sdio_interface();
            tries = tries + 1;
        }
    }
    if (global_sdio_driver == 0x0) {
        ok = 0;
    }
}
```
ABR: Identify code & data to remove, and remove!

- 574 functions removed
- ~144 KB of code removed
- Modified image passed OEM functional test suite with “performance consistent with the original firmware image”
  - ECM reflash of 7KB a second
  - Monitoring test 1261.35 parameters per second

Graphical representation of ABR applied to the VDA firmware, removing wireless communications. Light regions represent removed code regions, dark regions represent unchanged regions.
Cybersecurity Concerns & Mitigations

1. Disable debug interfaces from production devices
2. Reconsider Bluetooth and WiFi functionality
3. Secure device firmware
4. Perform runtime protection and monitoring
Contact Info

Wyatt Ford, Edward Larson
{wyatt, edward}@redballoonsecurity.com

Dr. Jeremy Daily
Jeremy.Daily@colostate.edu