Vehicle Diagnostics Adapter Cybersecurity Concerns with Wireless Connectivity

Edward Larson, Wyatt Ford,
Sam Lerner, Dr. Jeremy Daily

Red Balloon Security

SYSTEMS ENGINEERING
COLORADO STATE UNIVERSITY

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:

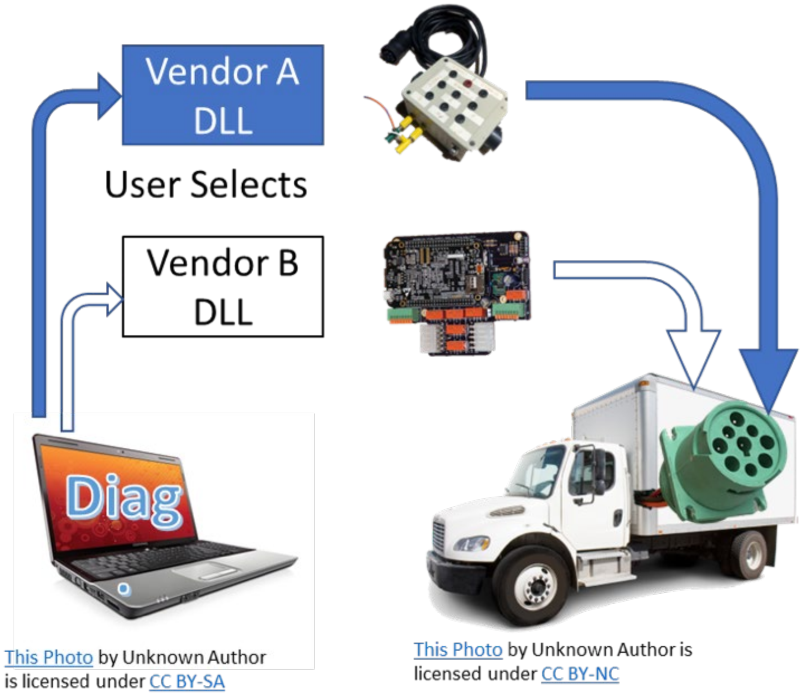
- Removes unwanted feature software: Bluetooth, WiFi
- Firmware with removed features has no unwanted side effects

Broader Goal

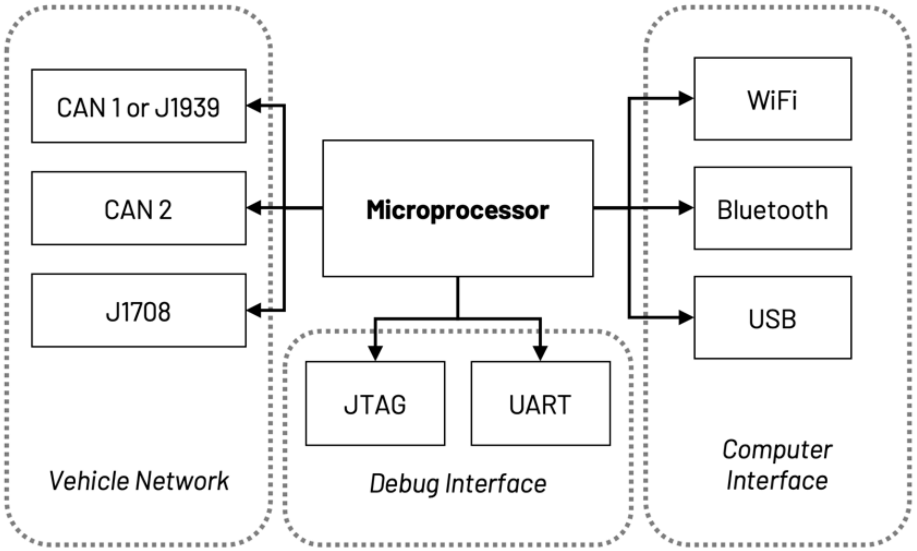
Create automated process to audit and harden *Component A*

- 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)?


Vendor A DLL
User Selects
Vendor B DLL
Diag
This Photo by Unknown Author is licensed under CC BY-SA
This Photo by Unknown Author is licensed under CC BY-NC

System Design

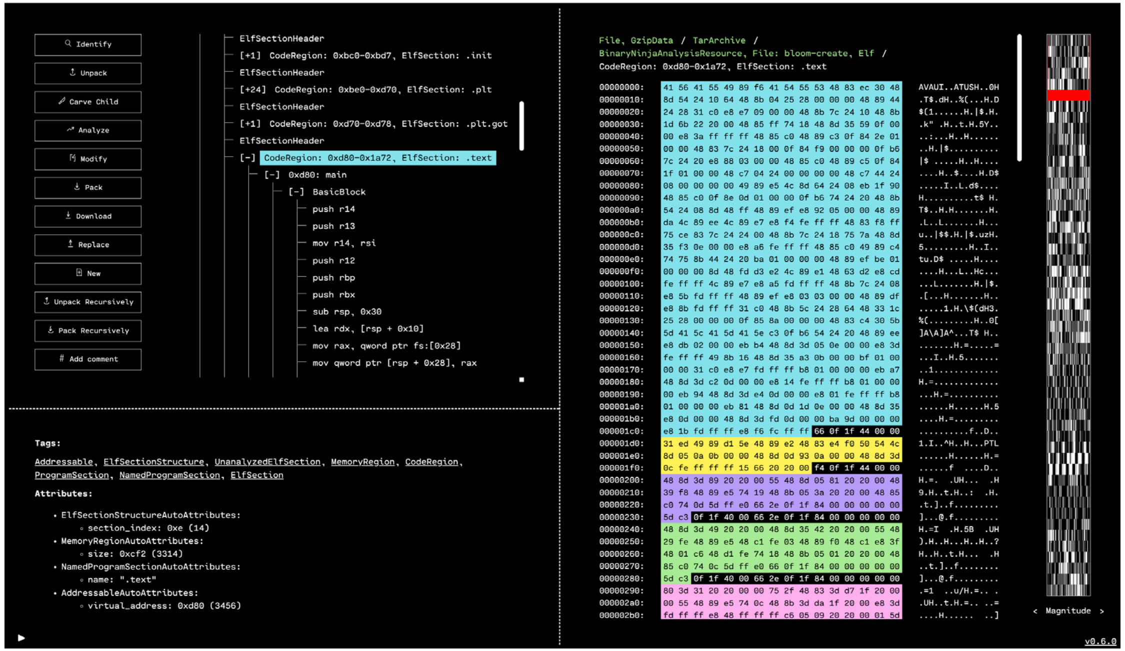


VDA Component Diagram

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)



```
├─ ElfSectionHeader
├─ [+1]  CodeRegion: 0xbc0-0xbd7, ElfSection: .init
├─ ElfSectionHeader
├─ [+24]  CodeRegion: 0xbe0-0xd70, ElfSection: .plt
├─ ElfSectionHeader
├─ [+1]  CodeRegion: 0xd70-0xd78, ElfSection: .plt.got
├─ ElfSectionHeader
├─ [-]  CodeRegion: 0xd80-0x1a72, ElfSection: .text
│   ├─ [-]  0xd80: main
│   │   ├─ [-]  BasicBlock
│   │   │   ├─ push r14
│   │   │   ├─ push r13
│   │   │   ├─ mov r14, rsi
│   │   │   ├─ push r12
│   │   │   ├─ push rbp
│   │   │   ├─ push rbx
│   │   │   ├─ sub rsp, 0x30
│   │   │   ├─ lea rdx, [rsp + 0x10]
│   │   │   ├─ mov rax, qword ptr fs:[0x28]
│   │   │   ├─ mov qword ptr [rsp + 0x28], rax
```

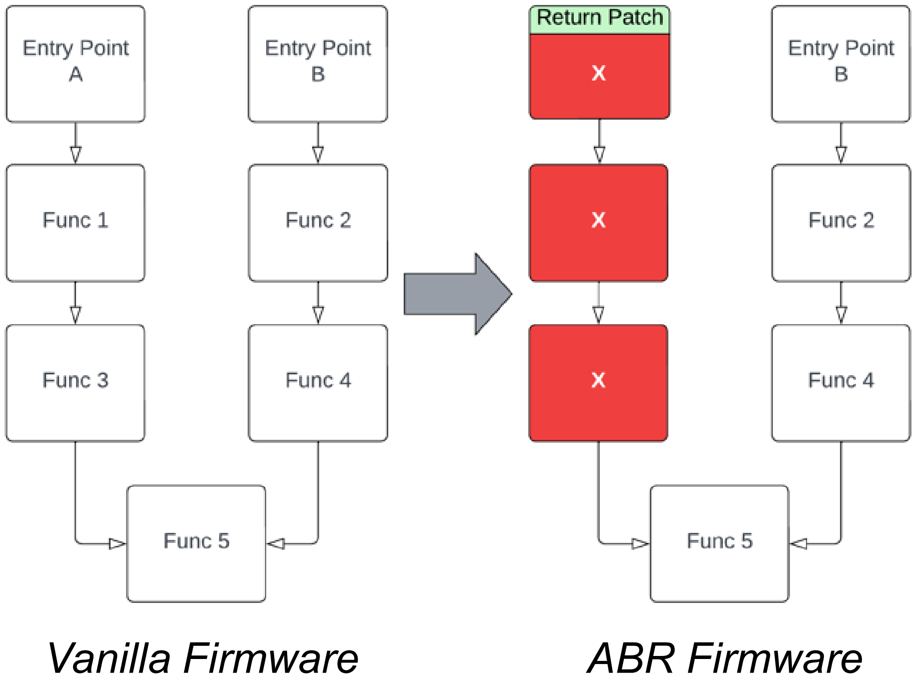
```
Tags:

Addressable, ElfSectionStructure, UnanalyzedElfSection, MemoryRegion, CodeRegion,
ProgramSection, NamedProgramSection, ElfSection

Attributes:

  • ElfSectionStructureAutoAttributes:
      ◦ section_index: 0xe (14)
  • MemoryRegionAutoAttributes:
      ◦ size: 0xcf2 (3314)
  • NamedProgramSectionAutoAttributes:
      ◦ name: ".text"
  • AddressableAutoAttributes:
      ◦ virtual_address: 0xd80 (3456)
```

```
File, GzipData / TarArchive /
BinaryNinjaAnalysisResource, File: bloom-create, Elf /
CodeRegion: 0xd80-0x1a72, ElfSection: .text

00000000:   41 56 41 55 49 89 f6 41 54 55 53 48 83 ec 30 48   AVAUI..ATUSH..0H
00000010:   8d 54 24 10 64 48 8b 04 25 28 00 00 00 48 89 44   .T$.dH..%(...H.D
00000020:   24 28 31 c0 e8 e7 09 00 00 48 8b 7c 24 10 48 8b   $(1......H.|$.H.
00000030:   1d 6b 22 20 00 48 85 ff 74 18 48 8d 35 59 0f 00   .k" .H..t.H.5Y..
00000040:   00 e8 3a ff ff ff 48 85 c0 48 89 c3 0f 84 2e 01   ..:...H..H......
00000050:   00 00 48 83 7c 24 18 00 0f 84 f9 00 00 00 0f b6   ..H.|$..........
00000060:   7c 24 20 e8 88 03 00 00 48 85 c0 48 89 c5 0f 84   |$ .....H..H....
00000070:   1f 01 00 00 48 c7 04 24 00 00 00 00 48 c7 44 24   ....H..$....H.D$
00000080:   08 00 00 00 00 49 89 e5 4c 8d 64 24 08 eb 1f 90   .....I..L.d$....
00000090:   48 85 c0 0f 8e 0d 01 00 00 0f b6 74 24 20 48 8b   H..........t$ H.
000000a0:   54 24 08 8d 48 ff 48 89 ef e8 92 05 00 00 48 89   T$..H.H.......H.
000000b0:   da 4c 89 ee 4c 89 e7 e8 f4 fe ff ff 48 83 f8 ff   .L..L.......H...
000000c0:   75 ce 83 7c 24 24 00 48 8b 7c 24 18 75 7a 48 8d   u..|$$.H.|$.uzH.
000000d0:   35 f3 0e 00 00 e8 a6 fe ff ff 48 85 c0 49 89 c4   5.........H..I..
000000e0:   74 75 8b 44 24 20 ba 01 00 00 00 48 89 ef be 01   tu.D$ .....H....
000000f0:   00 00 00 8d 48 fd d3 e2 4c 89 e1 48 63 d2 e8 cd   ....H...L..Hc...
00000100:   fe ff ff 4c 89 e7 e8 a5 fd ff ff 48 8b 7c 24 08   ...L.......H.|$.
00000110:   e8 5b fd ff ff 48 89 ef e8 03 03 00 00 48 89 df   .[...H.......H..
00000120:   e8 8b fd ff ff 31 c0 48 8b 5c 24 28 64 48 33 1c   .....1.H.\$(dH3.
00000130:   25 28 00 00 00 0f 85 8a 00 00 00 48 83 c4 30 5b   %(.........H..0[
00000140:   5d 41 5c 41 5d 41 5e c3 0f b6 54 24 20 48 89 ee   ]A\A]A^...T$ H..
00000150:   e8 db 02 00 00 eb b4 48 8d 3d 05 0e 00 00 e8 3d   .......H.=.....=
00000160:   fe ff ff 49 8b 16 48 8d 35 a3 0b 00 00 bf 01 00   ...I..H.5.......
00000170:   00 00 31 c0 e8 e7 fd ff ff b8 01 00 00 00 eb a7   ..1.............
00000180:   48 8d 3d c2 0d 00 00 e8 14 fe ff ff b8 01 00 00   H.=.............
00000190:   00 eb 94 48 8d 3d e4 0d 00 00 e8 01 fe ff ff b8   ...H.=..........
000001a0:   01 00 00 00 eb 81 48 8d 0d 1d 0e 00 00 48 8d 35   ......H......H.5
000001b0:   e8 0d 00 00 48 8d 3d fd 0d 00 00 ba 9d 00 00 00   ....H.=.........
000001c0:   e8 1b fd ff ff e8 f6 fc ff ff 66 0f 1f 44 00 00   ..........f..D..
000001d0:   31 ed 49 89 d1 5e 48 89 e2 48 83 e4 f0 50 54 4c   1.I..^H..H...PTL
000001e0:   8d 05 0a 0b 00 00 48 8d 0d 93 0a 00 00 48 8d 3d   ......H......H.=
000001f0:   0c fe ff ff ff 15 66 20 20 00 f4 0f 1f 44 00 00   ......f  ....D..
00000200:   48 8d 3d 89 20 20 00 55 48 8d 05 81 20 20 00 48   H.=.  .UH...  .H
00000210:   39 f8 48 89 e5 74 19 48 8b 05 3a 20 20 00 48 85   9.H..t.H..:  .H.
00000220:   c0 74 0d 5d ff e0 66 2e 0f 1f 84 00 00 00 00 00   .t.]..f.........
00000230:   5d c3 0f 1f 40 00 66 2e 0f 1f 84 00 00 00 00 00   ]...@.f.........
00000240:   48 8d 3d 49 20 20 00 48 8d 35 42 20 20 00 55 48   H.=I  .H.5B  .UH
00000250:   29 fe 48 89 e5 48 c1 fe 03 48 89 f0 48 c1 e8 3f   ).H..H...H..H..?
00000260:   48 01 c6 48 d1 fe 74 18 48 8b 05 01 20 20 00 48   H..H..t.H...  .H
00000270:   85 c0 74 0c 5d ff e0 66 0f 1f 84 00 00 00 00 00   ..t.]..f........
00000280:   5d c3 0f 1f 40 00 66 2e 0f 1f 84 00 00 00 00 00   ]...@.f.........
00000290:   80 3d 31 20 20 00 00 75 2f 48 83 3d d7 1f 20 00   .=1  ..u/H.=.. .
000002a0:   00 55 48 89 e5 74 0c 48 8b 3d da 1f 20 00 e8 3d   .UH..t.H.=.. ..=
000002b0:   fd ff ff e8 48 ff ff ff c6 05 09 20 20 00 01 5d   ....H......  ..]
```

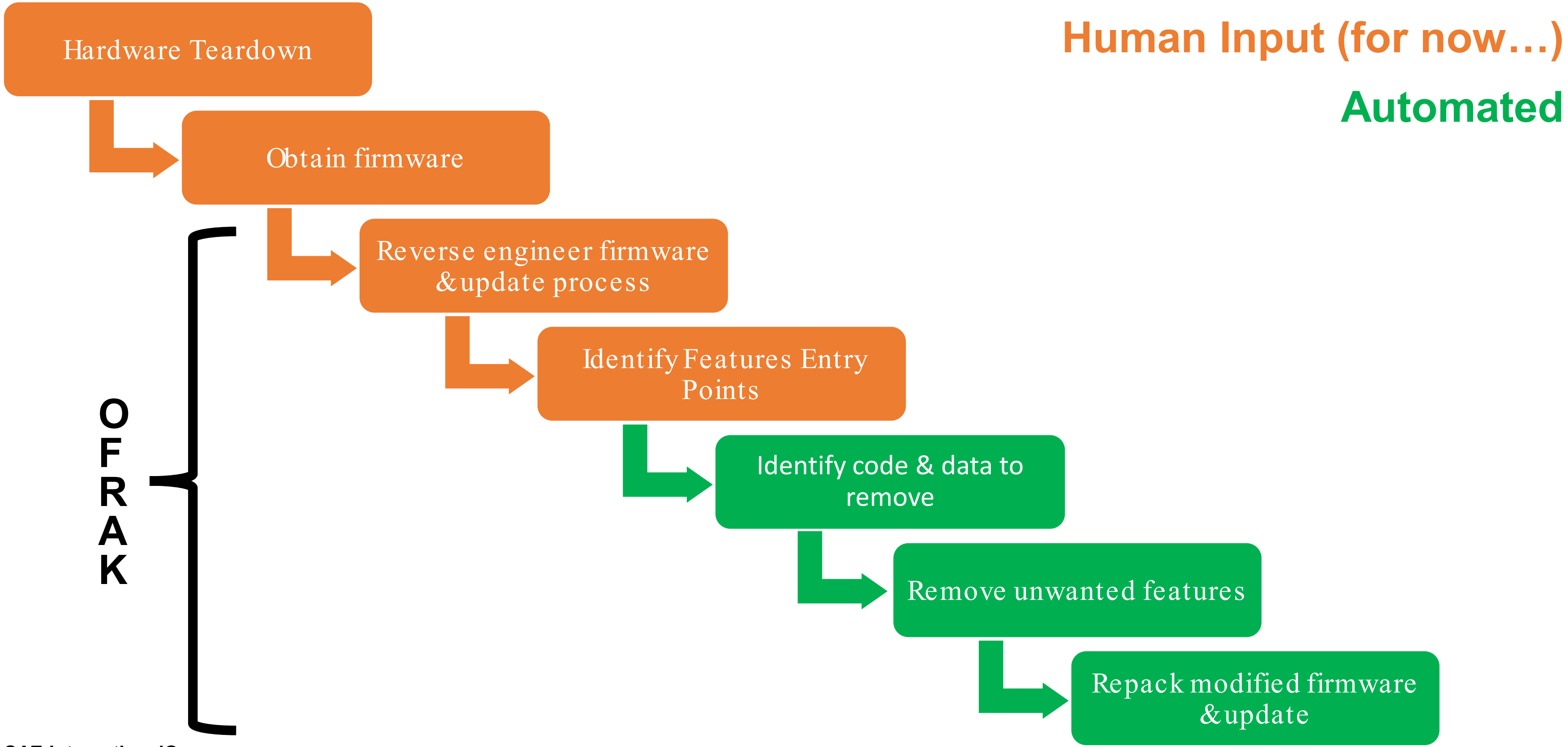
Autotomic Binary Reduction (ABR)

Automated removal of unnecessary or unwanted binary code and data

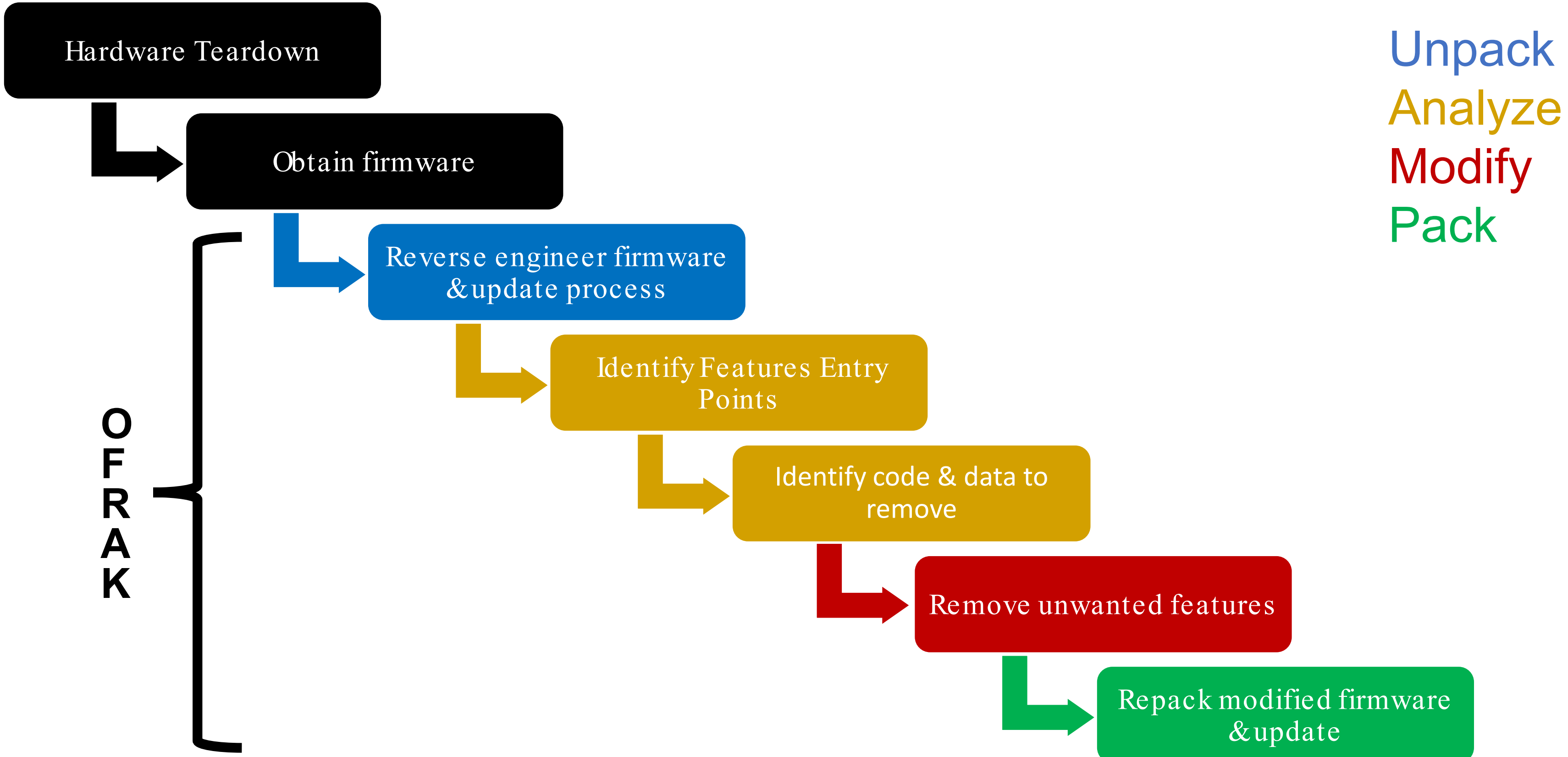
1. Identify **entry points**
2. Identify **code and data** *exclusively* control-flow dependent on **entry points**
3. Remove **code and data** from firmware
4. Replace with:
   a. Return patch
   b. Reclaimed space



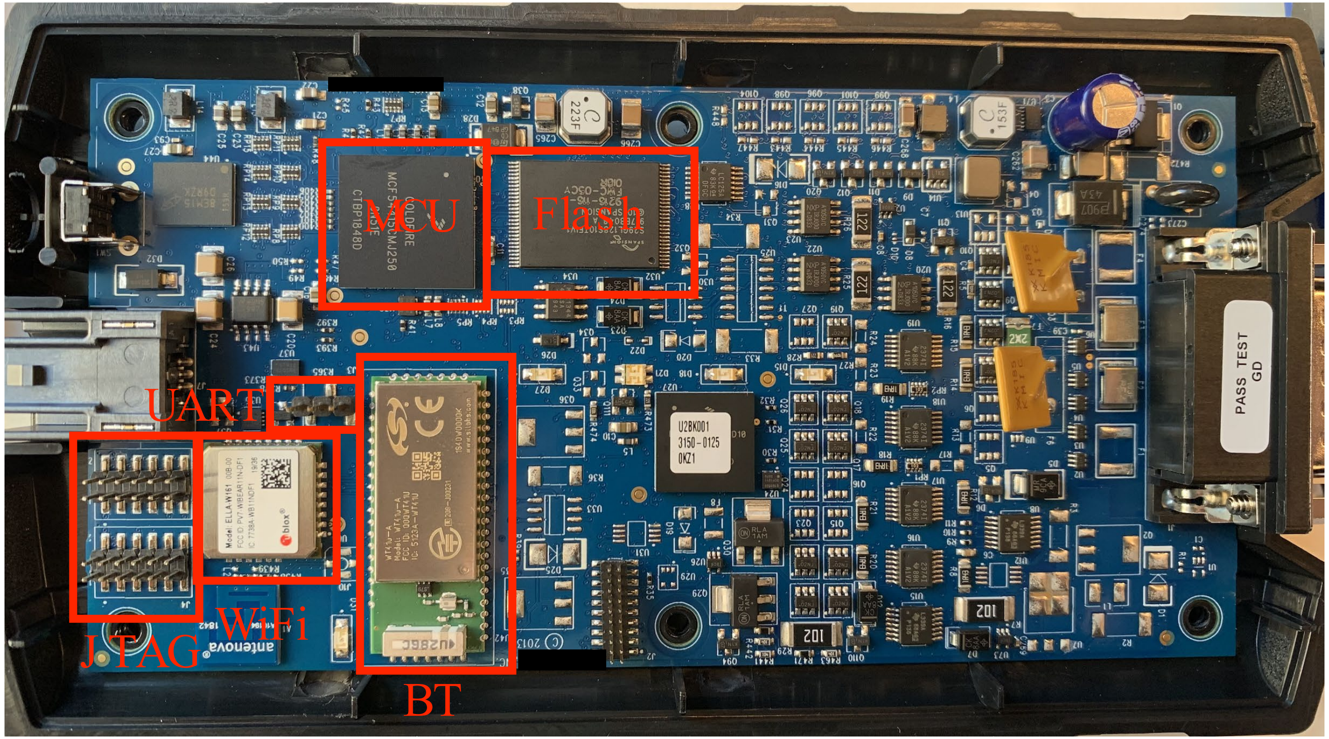
ABR Workflow


Human Input (for now…)
Automated
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
OFRAK

ABR Workflow in General OFRAK 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
OFRAK
Unpack
Analyze
Modify
Pack

Hardware Teardown


[redacted]
[redacted]

Our photo of the VDA after we opened the case, highlighting key components

Obtain Firmware, Reverse Engineer Firmware Update Process

Obtain Firmware

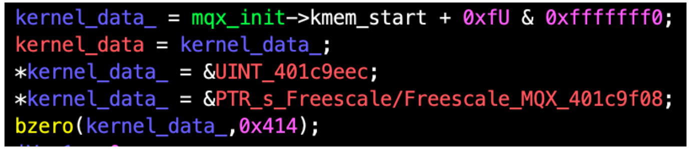
1. Extract firmware update file from Diagnostics tool
2. Use UART to discover memory mapping

Reverse Engineer Update using packet capture (PCAP)

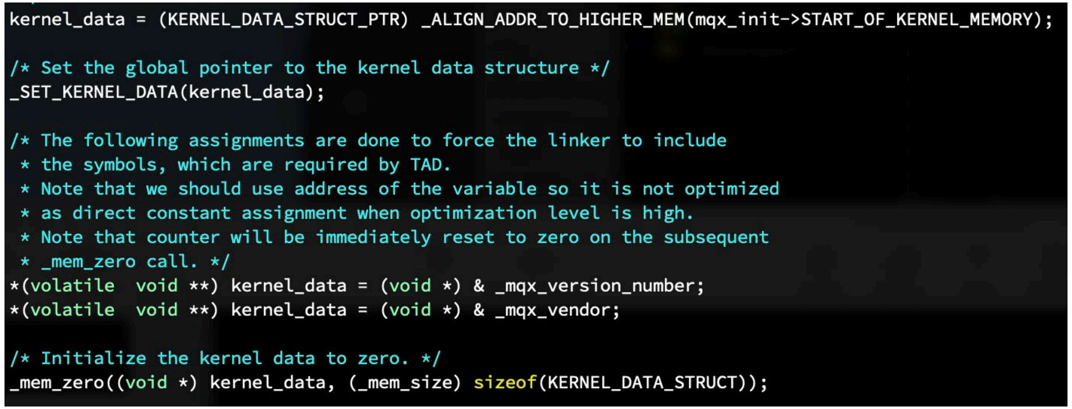
1. Send “\x03” + the size of the firmware update file
2. Send the firmware update file

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!



```
kernel_data_ = mqx_init->kmem_start + 0xfU & 0xfffffff0;
kernel_data = kernel_data_;
*kernel_data_ = &UINT_401c9eec;
*kernel_data_ = &PTR_s_Freescale/Freescale_MQX_401c9f08;
bzero(kernel_data_,0x414);
```

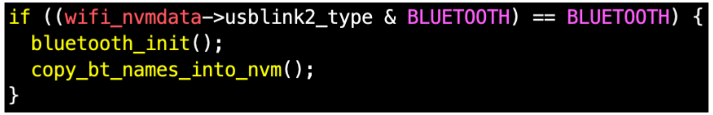


```
kernel_data = (KERNEL_DATA_STRUCT_PTR) _ALIGN_ADDR_TO_HIGHER_MEM(mqx_init->START_OF_KERNEL_MEMORY);

/* 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));
```

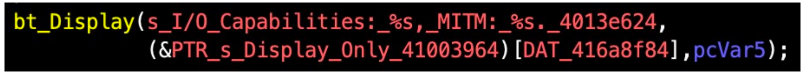
Identify Feature Entry Points

5 Generalized Entry Points to WiFi/Bluetooth subsystem

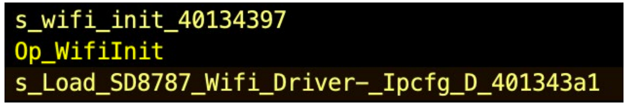
- OS tasks
- UART debug commands
- SDIO driver initialization code
- Marvell SD8787 driver code
- TI Bluetopia library code



```
if ((wifi_nvmdata->usblink2_type & BLUETOOTH) == BLUETOOTH) {
  bluetooth_init();
  copy_bt_names_into_nvm();
}
```

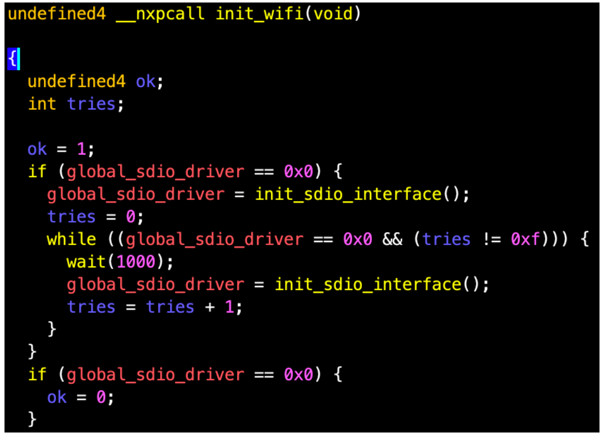


```
bt_Display(s_I/O_Capabilities:_%s,_MITM:_%s._4013e624,
              (&PTR_s_Display_Only_41003964)[DAT_416a8f84],pcVar5);
```



```
s_wifi_init_40134397
Op_WifiInit
s_Load_SD8787_Wifi_Driver-_Ipcfg_D_401343a1
```

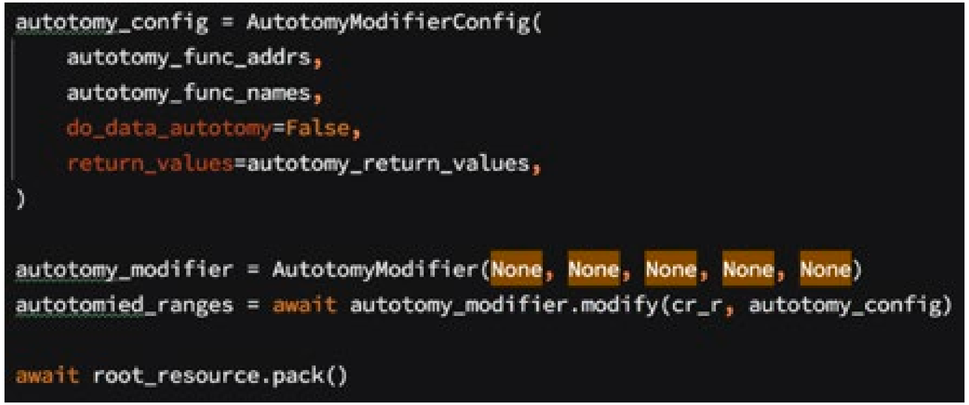
```
const char driver_version[] =
      "SD8787-%s-M2614" MLAN_RELEASE_VERSION "-GPL" "-(" "FP" FPNUM ")"
```



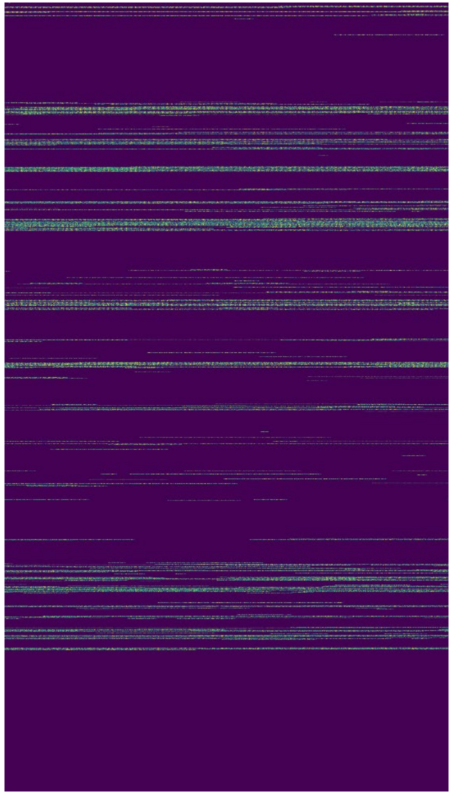
```
undefined4 __nxpcall init_wifi(void)

{
  undefined4 ok;
  int tries;

  ok = 1;
  if (global_sdio_driver == 0x0) {
    global_sdio_driver = init_sdio_interface();
    tries = 0;
    while ((global_sdio_driver == 0x0 && (tries != 0xf))) {
      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



```
autotomy_config = AutotomyModifierConfig(
    autotomy_func_addrs,
    autotomy_func_names,
    do_data_autotomy=False,
    return_values=autotomy_return_values,
)

autotomy_modifier = AutotomyModifier(None, None, None, None, None)
autotomied_ranges = await autotomy_modifier.modify(cr_r, autotomy_config)

await root_resource.pack()
```



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