Roadside to Everyone (R2E) Phase 1: Physical & Local Vulnerabilities in (C)V2X RSUs

AUTHOR: Jon Gaines - GainSec
<phrack@gainsecmail.com>

Table of Contents

0. Introduction

Roadside Units (RSUs) play a critical role in Vehicle-to-Everything (V2X) communication, guiding autonomous vehicles, managing traffic flow, and even assisting pedestrians. However, these units are often installed in public locations with minimal physical security — the units I have are secured by just four screws — making them highly vulnerable to physical attacks. Unlike traditional cyber threats that require remote exploitation, an attacker with brief physical access can stealthily gain persistent complete control without any need for soldering or physical modifications. Once compromised, an RSU becomes an invisible tool for manipulating V2X systems, with potentially catastrophic consequences.

The risks are not theoretical, as more and more RSUs are deployed worldwide. Past hacks on roadside LED signs, often left unprotected, have shown how easily public infrastructure can be tampered with. However, while a hacked LED sign might display prank messages, a compromised RSU poses far greater dangers. Attackers could mislead autonomous vehicles, disrupt traffic signals, or even manipulate pedestrian safety systems. For example, a visually impaired person relying on a V2X-enabled crossing alert could be given false information, stepping into traffic at the wrong time. In fact, the vendor of these RSUs has already developed an Android and iOS application that assists visually impaired folks with crossing the street safely [1]. Similarly, emergency

vehicles could be rerouted into congestion, or fake hazard warnings could create unnecessary panic and detours.

As smart transportation systems become more widespread, securing RSUs against stealthy local threats is no longer optional -- it is essential for public safety and the integrity of connected infrastructure.

This paper explores some of the physical and local vulnerabilities found in Kapsch TrafficCom RSUs, highlighting how easily they can be compromised and the far-reaching impact of such attacks. This paper presents results from phase 1 of a multi-phased approach to reverse engineer these units; the goal of Phase 1 was gaining root access. I'll also discuss what my plans are for Phase 2. I was not expecting to have seven CVEs published -- CVE-2025-25732, CVE-2025-25733, CVE-2025-25734, CVE-2025-25735, CVE-2025-25736, CVE-2025-25737, CVE-2025-25738. However, due to the low attack complexity and never hearing back from the vendor after multiple attempts to contact them, here we are.

Few things to note, I tried to keep general concepts/ explanations, that aren't unique to these devices, brief. I've included commands, outputs, reproduction steps, etc. as much as I could. As I never heard back from the vendor, there are a good chunk of things that I've left out, as I don't

want to be sued. That said, where applicable, I included some source code snippets, paths, binary names, etc. and hope to eventually upload all the custom drivers, scripts, source code, etc. that I found on these devices one day. Check out my site or reach out for screenshots or further information.

Cela dit, entrons dans le vif du sujet!

1. Overview of (C)V2X & its associated technologies

1.1 V2X & CV2X

Vehicle-to-Everything (V2X) is a communication system that enables vehicles to interact with various elements in their environment, including other vehicles (V2V), infrastructure (V2I), pedestrians (V2P), and networks (V2N).

This interaction aims to enhance road safety, improve traffic efficiency, and support autonomous driving by facilitating the exchange of real-time information. Cellular Vehicle-to-Everything (C-V2X) is a specific implementation of V2X that utilizes cellular network technologies, such as 4G LTE and 5G, to provide two complementary communication modes. Often called Direct Communication & Network Communication respectively. The main thing to note for this paper and these units is that the CV2X units (9260) support the same V2X that the V2X units (9160) only support. [2] V2V, V2I, V2P, V2N cover entire huge varieties of communications such as Intelligent Traffic Signal Systems (ITSS), Interaction Geometry (MAP), Signal Phase and Timing (SPaT) and much more. [3] This is out of scope of Phase 1 though.

Although this technology isn't new, it's yet to be implemented on every corner worldwide. Below is a list of places where the RSUs I tested have been deployed (some RSUs might be the newest model 7360 [12] which I haven't found for sale on the 2nd hand market).

- Greeley, CO [4]
- DOT I-70, CO [5]
- Ohio [6]
- Port Bilbao, Spain [7]
- Ipswich, Australia [8]

1.2 DSRC & ITS-G5

Dedicated Short-Range Communications (DSRC) is a wireless communication technology specifically designed for vehicle-to-everything (V2X) applications. It operates in the 5.9 GHz band and enables low latency, high-reliability communication between vehicles (V2V), infrastructure (V2I), and pedestrians (V2P). DSRC is based on IEEE 802.11p (11p) -- a Wi-Fi variant for which you'll find old Atheros 9 Wireless Cards have a custom Linux driver that supports sniffing -- and has been widely used for safety-critical applications like collision avoidance, traffic signal priority, and road hazard warnings. Compared to Cellular V2X (C-V2X), DSRC has a shorter range and does not rely on cellular networks but are limited in scalability and future compatibility with 5G advancements. DSRC, however, is what is widely used in the USA at least at the time of writing.

At the protocol level, which I'll expand on in a later phase, Wireless Access in Vehicular Environment (WAVE) is utilized.

ITS-G5 is the current primary European standard for V2X. For simplicity of this paper, you can consider ITS-G5 a variant of DSRC as it's based off DSRC and 11p. The RSU units I tested support both DSRC and ITS-G5. [9]

1.2 RSU & OBU

A roadside unit (RSU) and onboard unit (OBU) are communication devices that facilitate V2X communication. They are installed roadside and in-vehicle (who would've thought) respectively. In Phase 1 I have not covered OBUs, but I have acquired some and will start to test them soon. There are only a few large vendors of these devices and since they are used in critical infrastructure (at least in the case of RSUs) you will find many product pages with no price, little to no documentation, and certainly no SDKs if you can't provide a picture of a unit you own or at least a serial number.

2. Tested Devices

2.1 Models

I originally planned to only test 1 9160 and 1 9260. However, over the course of many trials and tribulations, I ended up soft bricking, frying or otherwise running into situations where having more devices was needed or ideal (copium). So logically I ended up with 6 units, although 2 are in a somewhat broken state. If you're interested in what I did, feel free to reach out, as long as you don't tease me.

Anyway, the model numbers of my units are: RIS-9160-1A0W [10] and RIS-9260-1AEW (Which seems to be the basic configuration) [11]. In reality, the main difference between the two units is that the 9260 comes with an LTE modem (+ bridge board), and a configured wireless stack.

FCC has some useful docs for these units, but they have been approved for a permanent confidentiality request so even the "technical manual" isn't super useful. Still worth checking out, but note that the unit shown in the pictures seems to be a prototype or earlier/revision and the bridge board has way more functionality than the units I've encountered in the wild. [13]

2.2 Chipsets & Notable In-Circuit Components

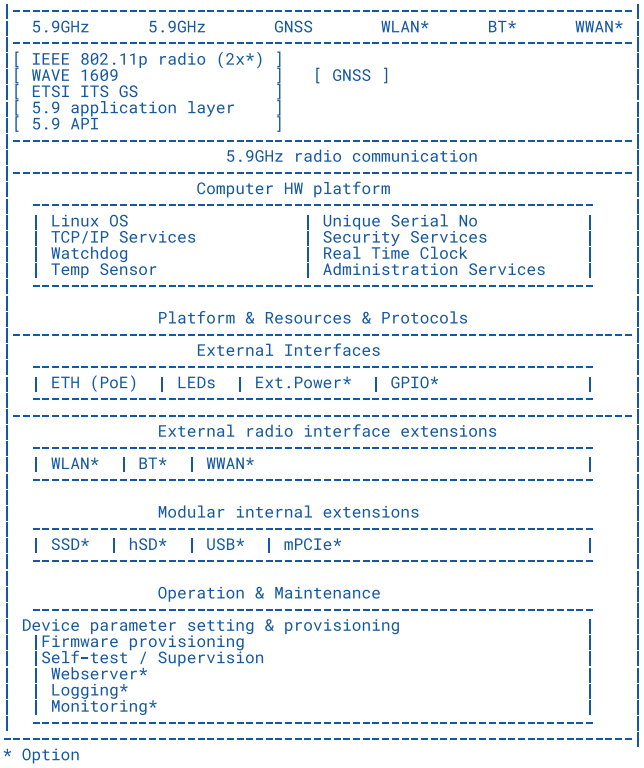
The main difference between the 9160 and 9260 is the addition of an LTE module via bridge board (even the firmware/OS state of the 9260 is often the same as that of the 9160). What's nice about this is that I don't have to list a ton of different components.

Computer-on-Module:
COMe-mBTi10 (Here's a datasheet)[14]

CPU: Intel E3825
RAM: 1GB
SPI Flash Memory/EEPROM:
Macronix MX25L6406E 8MB
eMMC: 4gb (3.6gb accessible)
RF: ALPS UMPZ2-ES4.1 (ALPS UMZ2 V753)
(User Guide for a similar chip: [15])
Hardware Monitor: Nuvoton NCT7802Y
Ethernet Controller: Intel Ethernet Controller I210-AT
GNSS: uBlox NEO-M8N-0-10
LTE-Modem: QualComm

I've included the block diagrams from the data sheets [10]

Here's the block diagram for 9160:



Here's the block diagram for 9260 [11]:

2.3 Notable Physical Interfaces

The 9160 and 9260 have the following notable physical interfaces:

- a microSD slot
- a mSATA slot
- 4 pins for read-only UART labeled "debug"
- 4 pins for I2C/SPI labeled "I2C"
- USB B Port (which can be configured for client or host via BIOS)
- a PoE RJ45 port
- some other goodies: there's another spot if you wanted to add an additional radio and another set of pads labeled "USB."

Unique to 9260 are the following physical components/interfaces:

- a bridge board with another set of read-only UART pins labeled "debug"
- an LTE modem with a dip switch
- a micro-USB port
- a 3 pin and 4 pin set

Additionally, the 9260 has a second RJ45 port, but I haven't determined its use yet.

2.4 Notable Firmware/Software

LEO Versions: 3.2.0.829.23, 3.8.0.1119.42, 4.6.0.1211.28

Embedded OS (EOS) Linux Versions (They're based off ELBE/Debian):

9260: 3.16.35-c3p+, 9160: 4.9.124-ris-152.29

Phoenix Secure Core: MVV1R976 X64 & MVV1R994 X64

V2X RSU Server: roadside

V2X Chipset Firmware Version: 1.19.0-deb9-x86 1002

LTE Modem Firmware: MDM9650

Other relevant binaries: acme (V2X radio test), llc (same thing but when using the Cobra Wireless binary [cw-llc]), rsu-configuration (custom V2X configuration script), cv2x-daemon & cv2x-config, C2X_HOST_APP.bin (API wrapper to interact with SXF17XX), aerolink libraries, v2x_radio_test, mm_server

I also included a list of custom debs found on the units in the raw output section of this paper.

3. Attack Surface

3.1 Lack of Everything

To be blunt, there is a lot to this embedded device, and to the V2X/CV2X stack, and since the device isn't on every corner or in every car, there isn't much public information about it. There were some vulnerabilities found in IPA utilized and I've covered the state of public vulnerabilities, teardowns, explanations or any other kind of information (easily) accessible to the public. Due to this, the entire hardware side of vehicle-to-everything, in its current state, is in its lack-of-everything. Hopefully my work -- however basic it is -- will get some other hackers jumping down these rabbit holes with or without me.

In the next few sections, I'm not going to list all the types of vulnerabilities that you can find within each category but more just to give a starting place to do your own research into them.

Most of the vulnerabilities from Phase 1 do not cover all the following categories so I'm opting to exclude going into vast detail. As I continue with my research, I will write more about them.

3.2 Hardware

Obviously with so many physical components and interfaces, there is a ton of attack surface. Whether it be attaching a SOIC clip to the SPI chip to overwrite the firmware, modify the EEPROM, read/modify over I2C, insert malicious drivers or boot off USB, mSata, or microSD. Not going to list it all here, but the physical/hardware attack surface is extensive.

3.3 Pre-Embedded Operating System (EOS) (BIOS/Bootloader/Kernel)

The pre-EOS components are definitely a great place to start on the software/firmware side of things. There is some documentation on Phoenix SecureCore and obviously plenty on syslinux. There are some custom kernel modules but I haven't had to do much beyond modifying kernel modules to get root access in Phase 1.

3.4 Firmware/EOS

A bunch of the firmware and EOS is closed source, custom, proprietary, confidential, requires a license and is in no meaningful way in the hands of the public. That said, what I will state is, these units all run a customized Embedded Linux Build Environment (ELBE) Debian-based Linux EOS.

They often utilize a modified version of firmware if not running a completely custom firmware built from scratch. This ends up being so fascinating as it's all new and custom but also, it's so much work to wrap your brain around it all.

One example: I found one blog post about wardriving/sniffing V2X (DSRC) signals which utilized old Atheros ATH9K wireless cards with custom drivers and a great DefCon lecture [16] but that's it. Then I found an academic paper and a GitHub Repo -- which tracks which ATH9K Wireless NICs are supported. The academic paper and GitHub repo mention that, *in theory* some ATH10 wireless cards should work on the devices as well.

So it was then wild to find a custom ATH10 firmware on the devices.

This firmware is located at:

```
/mnt/c3platpersistent/opt/firmware/ath10k/QCA988X/hw2.0/
```

4 - Achieving Root

4.1 - Simple and Useful

These systems come shipped with an EFI shell (I know, right?) which makes things a lot easier to deal with. There is a lot you can do via the EFI shell but this section is going to cover the simplest path to gaining root access on the device.

Unscrew the four screws to pop the cover off the unit. Then attach your UART adapter to get output, insert your FAT32 formatted microSD card into the port, and insert a keyboard into the USB port. Press F5 to enter the boot menu upon connecting your PoE cable and select “Internal Shell”. Once in the EFI shell, you'll see that the microSD is mounted as fs0 and the system's emmc is mounted as fs1. Use the cp command to copy fs1: to fs0:.

Command:

```
cp -r fs1:\ fs0:\images
```

Unplug the PoE cable to shut off the unit, pop out the microSD and insert it into a Linux box. cd to the microSD and then the /live/ directory.

Now use unsquashfs -s rootfs to get information including compression and block size which are needed to resquash the EOS after making modifications.

Cool, now you can cd into squashfs and view the EOS files. By default, these units are configured for IPv6 exclusively, so at minimum you need to add a bridge to etc/network/interfaces such as the following:

```
auth eth0:1
iface eth0:1 inet static
address 192.168.1.XXX
netmask 255.255.255.0
gateway 19.168.1.1
```

Now, add your ssh public key to root/.ssh/authorized_keys and be sure to chmod the .ssh/ and authorized_keys files properly. Lastly, uncomment the authorized_keys line within the sshd configuration file.

Now resquash the modified EOS filesystem via:

```
mksquashfs squashfs-root output/rootfs -comp gzip -b 131072 -noappend
```

Copy the new rootfs to the microSD, etc. Once back in the EFI shell, cp the new rootfs to /live/rootfs. For good measure, mv the Manifest and Manifest.asc files and rename them with .bak.

Now exit the EFI shell, wait for the unit to boot and:

```
ssh -i root_priv_key root@192.168.1.XXX.
```

If you're prompted for a password, it's likely that the ssh server is asking for an outdated algorithm for the key, so use this to get around that temporarily:

```
ssh -o PubkeyAcceptedAlgorithms=+ssh-rsa -i root_priv_key root@192.168.1.XXX
```

Grats, you now have root access.

4.2 Quickest

To gain just any type of root access, you can just add a netcat backdoor to rc.local, cron.d, a custom service or some other ways that you know.

4.3 Troubleshooting

On some units I needed to disable IPV6 completely. There are a bunch of ways to do this such as via modifying kernel parameters in /live/LIVE or in syslinux.cfg by adding 'ipv6.disable=1' to the APPEND line. Alternatively, by adding the following lines to /usr/tmplt/etc/sysctl.d/99-sysctl.conf:

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

If you're still having trouble getting the unit to be accessible over IPv4, I have a few other tips:

- Remove 'quiet' from the APPEND line
- add 'console=ttyS0,115200N8'

There are a ton of other kernel parameters you can add to assist in troubleshooting.

Although this shouldn't be needed, here are some other changes you can make:

- comment out any lines in avahi-daemon.conf that mention ipv6
- add eth0:1 to not be managed by avahi.
- Remove the 'rescue_mode=shell' from the APPEND line in RESCUE.

There are two versions of the EOS that the unit includes: the rescue and live versions. It was a coin flip if the unit I got initially tried to boot to rescue or live. If it's booting to rescue, there are a few extra steps but really you can modify the bootloader parameters to boot to live.

If you're struggling to get the device to boot to the live firmware image, just use the edit command within the EFI shell to modify the RESCUE file:

- change the BOOT_IMAGE=/rescue/vmlinuz to BOOT_IMAGE=/live/vmlinuz
- change the value of root_sfs from root_sfs=/live/rootfs to root_sfs=/rescue/rootfs

Now, you really shouldn't have to do this but if you're still having trouble, you can look into modifying the update_ipconfig.sh script to get IPv4 configured properly.

[... Read the full version of this article online ...]

8 - References

[1] https://apkpure.com/kapsch-ewalk/net.kapsch.ewalk/versions

[2] https://5gaa.org/c-v2x-explained/

[3] https://github.com/usdot-fhwa-OPS/V2X-Hub

[4] https://www.wjbf.com/business/press-releases/accesswire/983795/kapsch-trafficcom-supports-colorado-connected-vehicle-safety-project/

[5] https://www.kapsch.net/_Resources/Persistent/7b221a05b49b2d630a46508667e0d52de5f2efe7/Reference_Factsheet_Colorado_DOT_I-70_Corridor_EN.pdf

[6] https://www.kapsch.net/_Resources/Persistent/59824d4e81dce9d1a902359261304f6cf8231654/KTC-CVS-Reference_Ohio_33-SMC.pdf

[7] https://www.kapsch.net/en/press/releases/ktc-20240702-pr-en

[8] https://www.traffictechnologytoday.com/news/connected-vehicles-infrastructure/kapsch-supplying-equipment-for-australias-largest-c-its-connected-vehicle-pilot-project.html

[9] https://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.02.00_20/en_302663v010200a.pdf

[10] https://di9mr54a05a64.cloudfront.net/api-mciaustralia.expoplatform.com/media/MTYxODI3MDY3NzYwNzRkOWQ1ZThlYTI%3D.pdf

[11] https://www.kapsch.net/_Resources/Persistent/3d251a8445e0bf50093903ad70b3dbed34dec7e7/KTC-CVS_RIS-9260_DataSheet.pdf

[12] https://www.kapsch.net/_Resources/Persistent/b60658fe131d82fe20c5389cdc2f6425064f4a88/KTC-CVS_RIS-9360_DataSheet.pdf

[13] https://fcc.report/FCC-ID/XZU9160/

[14] https://www.mouser.com/datasheet/2/965/come_mbt10_datasheet-3235566.pdf

[15] https://content.u-blox.com/sites/default/files/EVK-THEO-P1_UserGuide_%28UBX-15013939%29.pdf

[16] https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEF%20CON%2025%20-%20Woodbury-and-Haltmeyer-Linux-Stack-Based-V2X-Framework-Hack-Connected-Vehicles.pdf