Security of connected vehicles

David Berard and Vincent Dehors
david.berard@synacktiv.com
vincent.dehors@synacktiv.com

Synacktiv

Abstract. In the context of the Pwn2Own Vancouver 2022 and 2023 contests, the Synacktiv team looked into several embedded systems of Tesla vehicles. The goal of this event is to find and report impactful vulnerabilities and demonstrate realistic attack scenario.

Modern cars have more and more features and connectivity. The attack surface increase and now the cars are fully reliant on electronic technology as well. Therefore, the security of the car computers (ECUs) is taken seriously by car manufacturers.

Whereas we have been able to demonstrate successful remote exploitation of a Tesla cars in 2022 and 2023, this article shows how the modern architecture makes these attacks complexe and less impactful. The hardware and software architecture of Tesla vehicles will be described with a focus on the security implications of the design choices made by the manufacturer. This article is blue team oriented and will present generic security principles and state of the art hardening applied on embbeded systems.

This article additionally provides insights on how security researchers can obtain the firmware and gain testing capabilities for some critical components.

1 Introduction

As the automotive industry continues to produce increasingly connected vehicles, a new range of risks and security concerns arise. Some of these risks exist since years but are now taking a new dimension as car technologies evolve. As early as 2014, security researchers have been concerned by the automotive field [4] and the impact of connectivity in those product. One particularity of automotive security if that there are a wide range of attacker profiles, each exploiting different parts of the car.

One of the main risks with connected cars is car theft, which has always been a concern even for non-connected cars. But attacks on connected cars that enable theft can now be executed on a larger scale and with greater ease.

A recent study [3] revealed that there are real-world attacks using CAN injection to steal cars by connecting a device to the CAN bus at a convenient, easily accessible location.

Keyless entry systems are more and more common in modern vehicles. Therefore, car theft through relay attacks on this system is a common problem, and there are many public researches on that.

Another important risk is the vehicle safety, car components are connected together through various CAN buses. By gaining access to these buses through hardware modification or by software attacks, attackers may affect the safety of the vehicle and cause people injuries or material damages.

The infotainment system has become one of the main component of modern cars, its screen provides many features: car control, internet connection, access to many external services. Like smartphones, this system contains many personal data: accounts credentials, connection tokens, browsing history, navigation history and even tokens to open the owner's garage. Gaining access to these data can facilitate attack that extend beyond the vehicle itself, and can also be used to track the vehicle position or for espionage activities.

Modern cars like Tesla ones also have many limitations enforced by software. For example, some features like advanced autopilot can be purchased during the vehicle life and do not require hardware modification or going to a service center. Bypassing these limitations (hardware and software) has always been the passion of some people, and they are sharing their findings to a large community. That kind of modifications (or attacks in some cases) must be taken into account by the car manufacturer in its security model.

2 Security and hardware design

This chapter focuses on the different components embedded in the Tesla car computer bundle containing the Infotainment system.

This package contains several PCBs:

- The board containing the Infotainment system
  - Intel or AMD SoC
  - Memories containing firmware and user data
  - Audio system
  - Smart Ethernet switch
  - Security Gateway
  - WiFi/Bluetooth connectivity
- The board hosting the autopilots
- The connectivity card for LTE connection and emergency calls

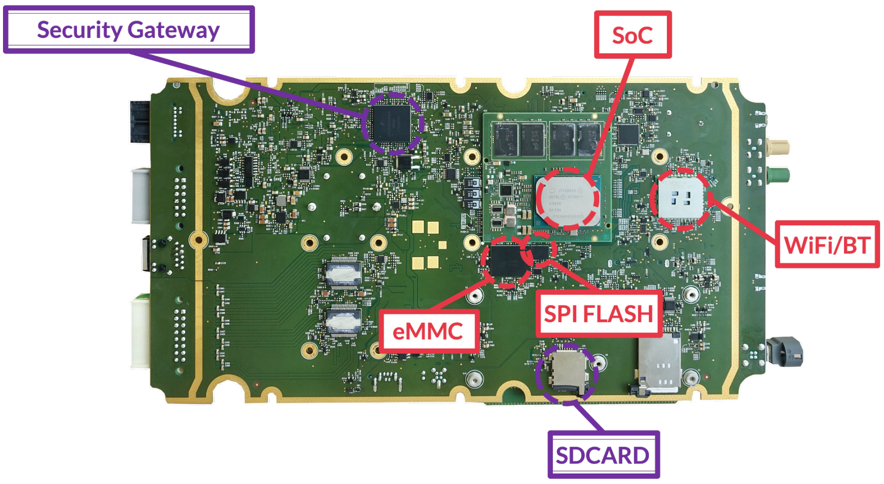


Fig. 1. Electronic board containing the Infotainment and the security Gateway

These different components are interconnected via Ethernet. Additionally, a diagnostic port located in the glove box allows direct connection to the switch.

2.1 Security from the hardware design

The security is considered from the hardware design, with most components chosen to ensure certain security features (Secure boot, encryption,

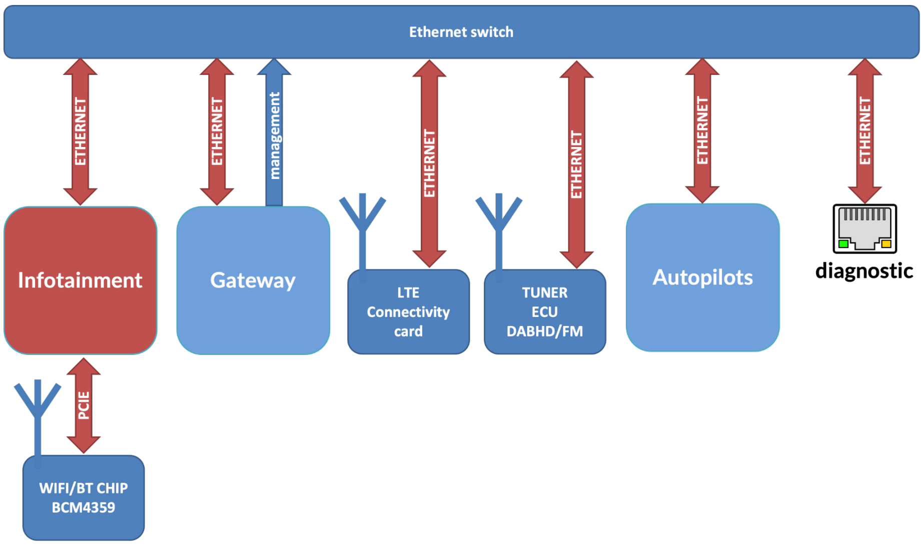


Fig. 2. Hardware architecture

filtering, etc.). The design also takes into account the possibility of compromised ancillary components.

For example, links to external interfaces (e.g., LTE card or Tuner) are filtered at multiple levels, and one of these levels is the Ethernet switch, which has filtering tables to allow only necessary communications for operation.

The main purpose of such architecture is to isolate critical components and make them harder to reach. The basic idea is to isolate the multimedia part of the car from safety-critical ECUs but modern architecture goes further by adding multiple layers of isolation and filtering. Of course, car manufacturers have to deals with other constraints like manufacturing costs or even physical constraints when they are designing the ECUs and their networks.

2.2 How several hardware versions are managed

Tesla has several hardware version for the Infotainment and autopilots boards, even on the Tesla Model 3. These boards are upgraded over time. Depending on the production date, the electronic components differ.

The Infotainment board was initially based on an Intel System On Chip (SoC) and has undergone several revisions. Today, new versions come with an AMD Ryzen SoC, but the system architecture remains very similar to the Intel SoC-based boards.

Tesla is working to standardize the Infotainment+Autopilot ECU across different models, so the Model S, Model X, Model Y, and Model 3 now share the same hardware and software.

The autopilot section is based on ARM Nvidia SoCs and has also undergone several revisions.

The security Gateway, responsible for providing Infotainment access to the CAN via Ethernet, is embedded on the Infotainment PCB, and it is a PowerPC architecture chip present in all hardware revisions.

Ancillary components embedded on this PCB, such as the Ethernet switch and connectivity cards (WiFi/Bluetooth), may vary depending on the hardware versions. The rest of this document focuses on the components embedded on the ECU based on the Intel SoC.

The LTE connectivity card varies within the same hardware version and is an external card connected to Ethernet (and other buses) via an M.2 connector on the PCB.

Despites all these hardware versions, several good practices are followed:

- Even if the hardware changes, the overall architecture design is kept as much as possible.
- The software is shared between all these versions and even between different car models
- There are still software updates for older hardware versions. The cost of maintaining multiple versions is lowered by the fact that all cars have the same software architecture and code bases.

3 Infotainment

The infotainment system is the computer for the multimedia related features which also manage the User Interface. For example, this system controls the main touch screen. As there are a lot of features and user-controled inputs in this system, this is an interesting target for an attacker. Even if the infotainment is well isolated in the car network, its security is still important because:

- The infotainment can do legitimate actions on the vehicle that can be considered as important in term of security, for example opening the trunk.
- This system contains sensitive user data.
- Compromising this system allows an attacker to reach a new attack surface in other critical part of the vehicle, for example the Security Gateway.

This section shows how the design of the Tesla infotainment limit these risks and greatly increase the cost of impactful attacks.

3.1 External memories

The usage of complex processors or System On Chips (SoCs) requires additional external components that are normally included in micro-controlers like external RAM and ROM memories. Therefore, there are multiple memory chips in the ECU's PCB that can contains useful information.

Persistent data storage (ROM) is often the first target for a black box assessment because it could allow an attacker to:

— Extract the firmware: this is the software running on the ECU. With these data, one can understand how the system works by reverse engineering and start looking for vulnerabilities.

— Extract user or car data: persistent memory often hold secrets or sensitive data. Sometimes, manufacturers implement encryption to protect those data.

— Execute code in the ECU by writing (or replacing) the ROM chip with a modified software, it may be possible to execute custom code. Gaining a first code execution, even with physical access, is a huge help looks for vulnerablities, debug and develop exploitation programs. To prevent this risk, manufacturers can choose to implement a secure-boot which checks the authenticity and the integrity of executed programs.

On the Tesla Infotainment, the Intel SoC uses an external eMMC flash. The difficulty to extract the content of this kind of component depends on its physical caracteristics and technology. For example, BGA chips are harder to dump as the pins are not physically available without unsoldering the chip. For the Pwn2Own 2022 contest, we had to keep the Infotainment working so the eMMC has been extracted without being unsoldered by communicating with the eMMC after the Intel SoC booted. To prevent from voltage conflict on the MMC signals, the SoC has been put in a alternative boot mode in which it does not use the eMMC. A Linux-based small computer (SBC Beagle Bone Black) has been used to communicate with this eMMC using the SDIO protocol allowing to extract and write the content of this memory.

When dumping high-speed bus like for eMMC, a clean hardware setup is required to prevent from electromagnetic signal perturbations. As our setup involve long aerial wires, our sdio driver has been patched to use the lowest frequency for this protocol.

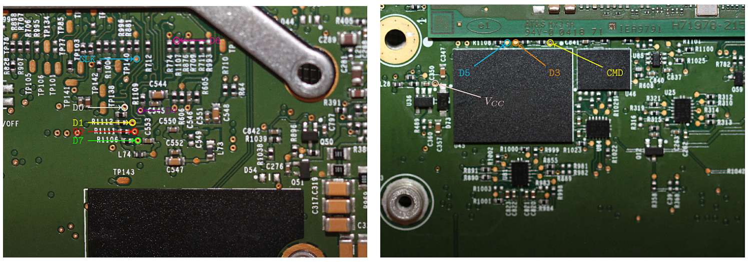


Fig. 3. Location of the eMMC signals on the PCB for the in-circuit dump

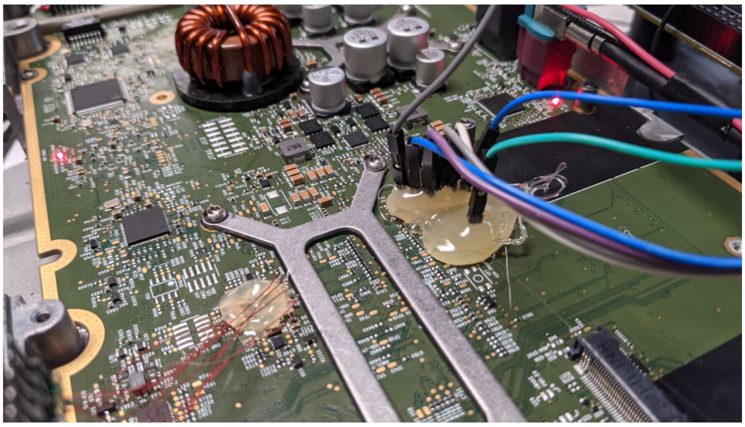


Fig. 4. Setup to connect the SBC for eMMC communication

With the ability to read and write the eMMC, an attacker can analyze its content but also can try to insert modified content. There are often bugs that bypass the secure-boot if the attacker is able to write the ROM memory. However, we did not find any on the Tesla model 3.

3.2 Secure boot and data encryption

The eMMC dump allows to understand how the infotainment boots and how the software integrity is checked.

The analysis of the partition table shows the following scheme:

— Partition "boot" (130MB): contains the Linux kernel and an initrd in two files (bank A and B)
— Partition "rootfs-a" (2GB): contains all the applications in a SquashFS filesystem
— Partition "rootfs-b" (2GB): same data as for "rootfs-a"
— Partition "lvm": a LVM volume which contains several partitions:

- Maps
- Games and game data
- Home (data of the infotainment applications) - **Encrypted**
- Log and Var (configuration and logging files) - **Encrypted**

The update uses a A/B scheme: the Linux kernel and the rootfs are written into the bank that is not currently used, and then the system reboots on the new version by toggling active slots.

Sensitive data like personal user data and credentials used by the Tesla services are stored in LVM encrypted partitions. The encryption key is protected by the hardware and is unique per vehicle. To extract this key, one needs to first execute (privileged) code on the Intel SoC.

The code integrity and authenticity is checked by a state of the art secure-boot. Secure boot root key is programmed on the hardware and cannot be extracted from the Intel SoC. They are used to check the bootloader and start the chain of trust: each next software component is checked:

- The bootloader is signed and verified.
- The Linux kernel is checked by the bootloader. The signature encapsulates the initrd which is embedded in the kernel binary.
- The initrd mount eMMC partitions and use **dm-verity** on the rootfs SquashFS partition (which is also read-only)
- All the executables are located in this SquashFS and data that are not in this partition are considered untrusted.
- Games are a special case: they are packaged as SquashFS files and have their own **dm-verity** configuration.

To sum up, the secure boot feature is well implemented and ensure that only authenticated software is started on the Infotainment. The confidentiality of sensitive data is assured by the encryption and the impossibility of executing untrusted code.

It is worth noting that there is nothing that manage automatically the end of live of these sensitive data. For example, we bought multiple ECUs on eBay which were probably coming from accidented vehicles. On each, data from the previous users were still present and not deleted even after powering up the Infotainment system. Some data like the history of navigation or saved accounts were available directly on the graphical interface. One of them even had a credit card information saved.

3.3 System and hardening

The Infotainment system is based on Linux. This is a Buildroot distribution heavily customized by Tesla. All system applications are stored in

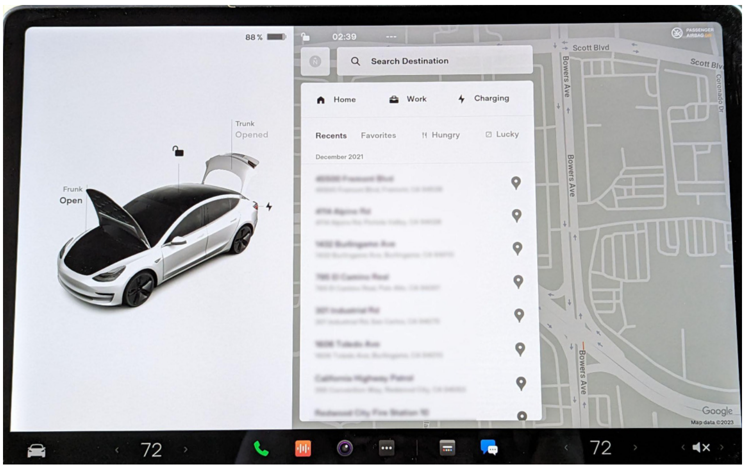
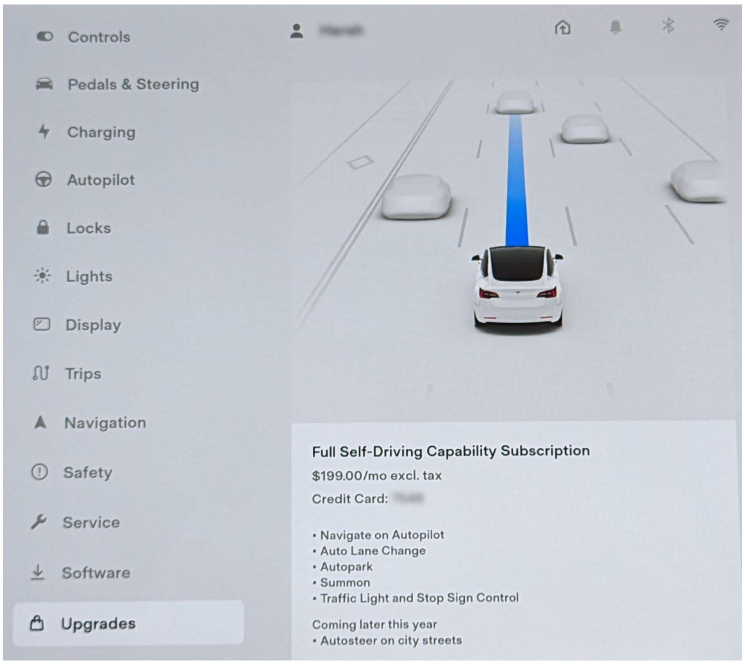


Fig. 5. History of navigation of the previous user



[redacted]

Full Self-Driving Capability Subscription
$199.00/mo excl. tax
Credit Card: [redacted]

- Navigate on Autopilot
- Auto Lane Change
- Autopark
- Summon
- Traffic Light and Stop Sign Control

Coming later this year
- Autosteer on city streets

Fig. 6. Credit card of the previous user used to subscribe to Tesla services

the SquashFS partition (read-only and protected by dm-verity). The init system is called "runit" and launches a lot of Tesla services directly after the boot like the graphical interface applications.

The design features a well thought in-depth defense which relies on several principles:

- Limit the attack surface
- Isolate and limit the applications rights
- Make vulnerabilities harder to exploit

The kernel configuration is a good example of attack surface limitation. Indeed, only the useful options are enabled and everything else is disabled directly from the compilation. This very light configuration limits the attack surface for remote and local attack. In order to make the exploitation harder, the configuration also includes all hardening options: KASLR, hardened allocator (hardened freelist, random freelist, ...).

The userland applications have multiple security mecanisms. First, each service runs with its own UID, with its own files and cannot interact easily with other processes or with the system. The binaries are often compiled with hardening options like using ASLR (PIE), having stack cookies and fine-tuned mapping protections. However, there is no CFI (Control Flow Integrity) present in the binaries.

3.4 Sandboxes

An important part of the defense in depth strategy relies on process isolation through the use of sandboxes:

- Strong network filtering with iptables using rules based on process UID. There is a whitelist for each outgoing connection for each service, and the default rule is to deny the network output flow.
- Filtering of syscalls with Kafel (seccomp): only syscalls normally used by the process are allowed. A malicious payload injected in the program will not be able to use other system calls, greatly limiting its attack surface for privilege escalation or exfiltrating data.
- Usage of the LSM (Linux Security Module) Apparmor: a profile is defined for each application, which restricts its network usage and applies a whitelist for all file accesses (including special files like drivers). Therefore, Apparmor also filters which other programs can be executed by the sandboxed application.
- Usage of minijail to configure Linux namespaces for isolating processes, filesystems, and network stack. For example, an application which should not use the network will be isolated in a dedicated empty network stack.

Not all programs have all the sandboxes. Some programs are not sandboxed at all, while others, such as games or web browsers, use all the isolation mechanisms. Sandboxes are very effective to limit the impact of a compromise and may be a unavoidable solution if the manufacturer needs to include less trusted third-party software. Services that have a remote attack surface should be sandboxed first as they are likely to be the entry point in the system.

The configuration of these sandboxes is often based on whitelists: the default behavior is to deny and only legitimate actions of a program are allowed. For example, a service that does not have permission to communicate on the network will be denied both incoming and outgoing network packets by multiple sandboxes at once. If this service is only allowed to listen on a TCP port, then only that particular communication is authorized. Moreover, the rules of these sandboxes are very well detailled to forbid anything that is abnormal. For example, using Kafel for syscall filtering, argument values can be checked in the sandbox rule.

Escaping these sandboxes is very complex as each one limits the attack surface to escape from one of the sandboxes. If there is a vulnerability, to escape from a sandbox (for example, in the Linux kernel), only a few applications will be able to access it.

3.5 Updates

The Tesla teams constantly work to correct known vulnerabilities through regular updates. The OTA update system searches and downloads updates from the Tesla backend automatically. As the Infotainment (and other ECUs) cannot reboot when the user wants to use the car, the update application is only triggered when the user choose to apply the update through the user interface.

The update system has been entirely developed by Tesla and is not based on open source software. The binary managing the update, called ice-updater, is present on different ECUs and models of the manufacturer. It is an interesting target for an attacker because it is privileged (root, without sandbox) and offers a large surface from the network. However, it has also been heavily audited and uses security-oriented programming practices: all controllable sizes and inputs are checked multiple times and there is no dynamic allocation. These kinds of practices drastically reduce the risk of memory corruption bugs.

3.6 Infotainment attack surface

An Infotainment system has multiple interfaces that can be attacked including from various network connections (diagnostic Ethernet port, WiFi network, mobile connection), the touch screen, USB ports, and Bluetooth. In competitions such as Pwn2Own, initial conditions are described in rules provided several months before the competition. There are two types of attack scenarios: those that do not involve any user interaction (zero-click) and those that require a user to be present in the car to perform manipulations (such as pressing the screen or plugging in a USB drive). Hardware attacks, however, are not allowed.

On Tesla cars, the attack surface accessible through the WiFi network is quite small. The Infotainment services do not listen on this interface, and firewall rules (Iptables) filter incoming and outgoing connections. Moreover, a significant portion of communication with Tesla servers goes through a dedicated encrypted tunnel established by a proprietary solution (Hermes).

The above architecture shows a robust and mature security architecture. However, in recent years, at least four successful attacks have been demonstrated on the Infotainment system: the team fluoroacetate at Pwn2Own 2019, T-bone [2] (2020), and our participations in 2022 [1] and 2023.

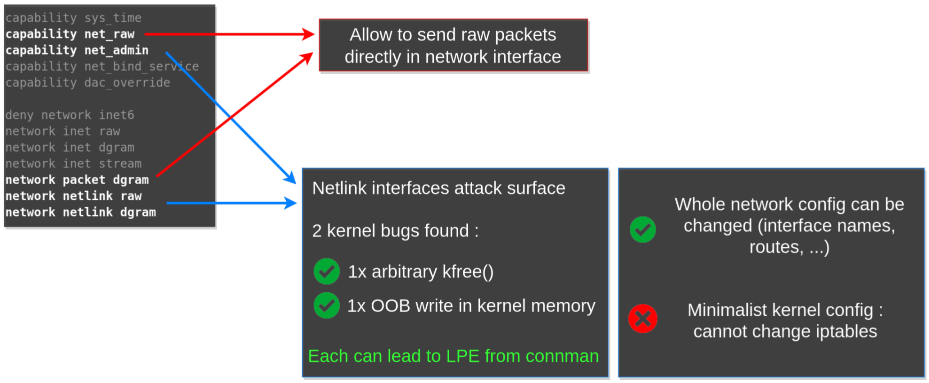
Two attacks targeted the same component: ConnMan, the service responsible for network management. It is open source and integrated by Tesla into their Buildroot distribution. One might think that if this software has been attacked twice in a row, it is due to poor code quality. However, few vulnerabilities have been identified in this service, and it has been sandboxed further after the T-bone [2] attack in 2020 and even further after ours.

The main reasons we targeted this software is that it is in a critical path of the architecture:

— It communicates with the outside world without authentication (DHCP, DNS, HTTP). This is one of the few surfaces accessible to an attacker from the WiFi network.
— The service needs to have some privileged rights because it manages the network.
— It is active without user action. For example, it automatically connects to "known" WiFi networks, making it part of the zero-click scenario. In addition, there is a known network for all Teslas, the Tesla Service network, whose credentials were obtained during the dump of the eMMC.

The amount of code in this software is quite small, and Tesla only compiles a small part of the open-source project, only the part necessary to configure a WiFi network and check connectivity with an HTTP request to their server. It reports the connectivity status and is controllable by another application via DBUS. That's how it communicates with the graphical interface.

During Pwn2Own 2022, only one memory corruption vulnerability (CVE-2022-32292) was used to obtain code execution in the context of this service. This vulnerability allows modifying the value of a byte (0x0A to 0x00) after the end of an allocation. Another double-free vulnerability (CVE-2022-32293) was also used to improve the exploitation time, but it is not necessary to obtain code execution. Thanks to the binary's protection mechanisms (mainly ASLR) and the reduction in surface in this service, the exploitation was extremely long and took several months of work. After obtaining a method of code execution in the context of Connman, it is only possible to perform actions that are normally allowed for this service due to the multiple sandboxes. But the service has two high-privileged Linux capabilities: CAP_NET_ADMIN and CAP_NET_RAW.



```
capability sys_time
capability net_raw
capability net_admin
capability net_bind_service
capability dac_override

deny network inet6
network inet raw
network inet dgram
network inet stream
network packet dgram
network netlink raw
network netlink dgram
```

Allow to send raw packets directly in network interface
Netlink interfaces attack surface
2 kernel bugs found :
1x arbitrary kfree()
1x OOB write in kernel memory
Each can lead to LPE from connman
Whole network config can be changed (interface names, routes, ...)
Minimalist kernel config : cannot change iptables

Fig. 7. Apparmor configuration for ConnMan

In the surface allowed by these sandboxes, Connman is able of using SOCKET_RAW because it needs to issue DHCP requests. This feature allows to communicate with other services and ECUs, which is normally not accepted by the network sandbox. But iptables rules do not apply to RAW sockets that inject packets into a lower-level layer in the Linux kernel.

During the competition, this bypass of iptables rules was used to communicate with the Secure Gateway and ask it to perform several actions on the CAN (such as opening the car's trunks, turning on the headlights and activating wipers).

Moreover, the capability CAP_NET_ADMIN offers a significant attack surface in the kernel. For example, ConnMan use the netlink API to communicate with the WiFi driver even if it is not an action it is supposed to do. Indeed, the control of the WiFi driver is done by the wpa_supplicant service, but there is no mechanism in these sandbox technologies to filter on the content of a netlink packet. Thus, a sandboxed program authorized to use netlink can perform all possible actions on this API. However, Tesla has removed a lot of kernel code thanks to a minimalist configuration. It is not possible, for example, to modify iptables rules from netlink. Two memory corruption bugs (CVE-2022-42430 and CVE-2022-42431) were found in the Wifi driver's netlink API. Each one allows for kernel code execution and thus bypassing all Infotainment protections (root without sandbox).

4 Ethernet network

4.1 Ethernet switch with filtering capabilities

The switch is a component on the PCB of the Infotainment system. In versions based on Intel SoC, it is the Marvell 88ea6321. This is not a simple Ethernet switch as it has the ability to filter packet depending on rules stored in a table called TCAM. The Security Gateway ensures the configuration of this switch via its MDIO bus. Numerous settings are available, allowing Tesla to implement filtering on the Ethernet network and to ensure that only the packets for normal operations pass through the switch.

Filtering is configured on all switch ports. Thus, from the diagnostic Ethernet port or even from a compromised component, the attack surface towards other components is significantly reduced or even non-existent.

The analysis of the switch configuration can be done in two ways:

— Reverse engineering of the Security Gateway firmware
— Analysis of MDIO frames with a logic analyzer

We used the second method, as it allows to dump a complete configuration from a startup (the configuration being applied by several software components of the Security Gateway).

The documentation for the Marvell switch is subject to an NDA and is not publicly available. The PINOUTs found in public documentation for switches of the same family do not match the one observed on the board. The location of the MDIO bus was found by probing the various tracks that seemed compatible with this signal.

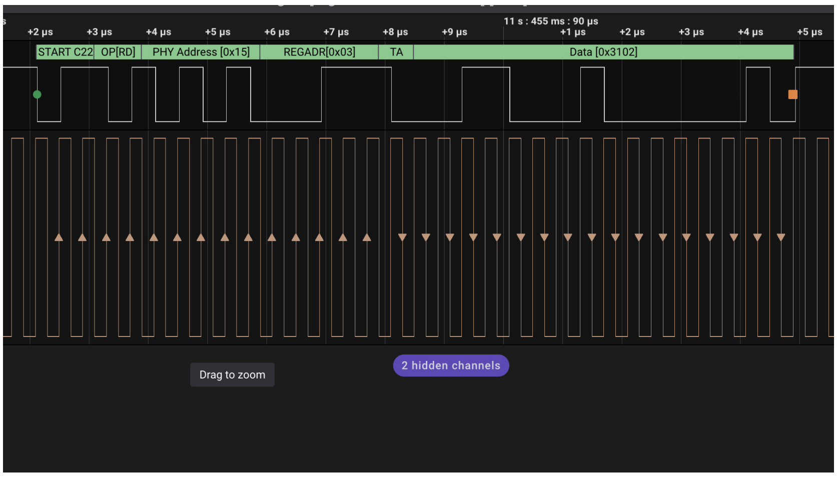


Fig. 8. MDIO bus signals

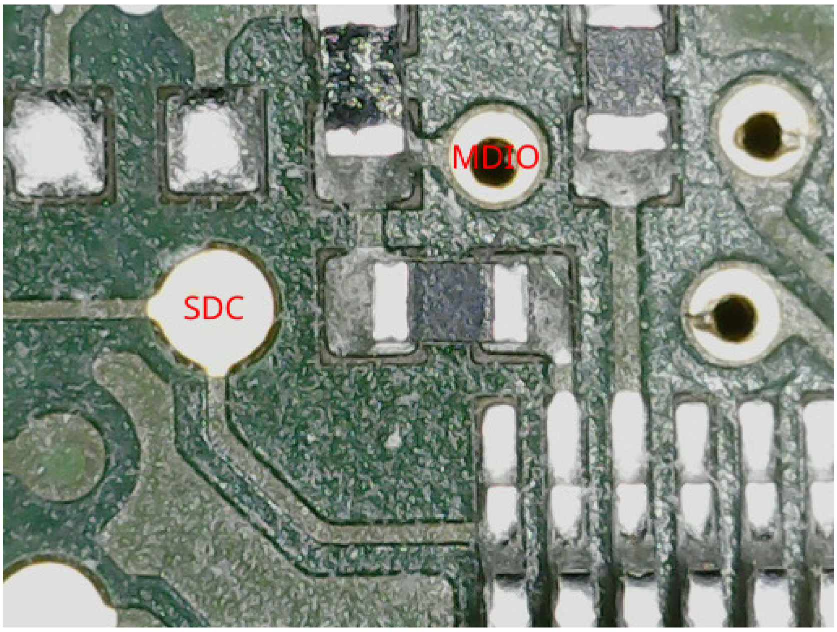


Fig. 9. MDIO bus position on the PCB (near the switch)

Although no public documentation describes the configuration protocol on MDIO, there is an implementation for Linux (drivers/net/dsa/mv88e6xxx). This can be studied to decode the various MDIO register reads and writes.

4.2 TCAM

Filtering is ensured by the TCAM entries of the switch, decoding MDIO frames allows to reconstruct these entries and obtain the filtering rules.

```
tcam entry    0: src_port=3, dst_port=0, eth_type=0x0800,IPv4,TCP,
    tcp_dport=22,
[...]
tcam entry    2: src_port=3, dst_port=0, eth_type=0x0800,IPv4,TCP,
    tcp_dport=8080,
[...]
tcam entry    4: src_port=3, dst_port=0, eth_type=0x0800,IPv4,TCP,
    tcp_dport=8081,
[...]
tcam entry   38: src_port=3, dst_port=DROP ip_src=192.168.90.60/32
tcam entry   39: src_port=3, dst_port=DROP ip_src=192.168.90.100/32
tcam entry   40: src_port=3, dst_port=DROP ip_src=192.168.90.103/32
tcam entry   41: src_port=3, dst_port=DROP ip_src=192.168.90.105/32
tcam entry   42: src_port=3, dst_port=DROP ip_src=192.168.90.104/32
tcam entry   43: src_port=3, dst_port=DROP ip_src=192.168.90.102/32
tcam entry   44: src_port=3, dst_port=DROP ip_src=192.168.90.30/32
[...]
```

Filtering is done by physical port. Rules are applied for each one to restrict the source IP address. In the example above, rules 38 to 44 prevent from using the IP of an internal component from the diagnostic port (port 3).

Packets that do not match any rule are not relayed by the switch (with the exception of port 0).

We can see in the example above that only TCP ports 22, 8080, and 8081 are allowed from the diagnostic port.

4.3 Switch reconfiguration for testing purposes

Due to the precise filtering provided by the switch, it is not possible to use the diagnostic Ethernet port for component testing. For example, communication with the Security Gateway is not possible from this port.

However, with hardware access, one can reconfigure the switch by communicating on the MDIO bus instead of the Security Gateway and disable all filtering. For the Pwn2Own contest, we adopted this method to be able to communicate with ECUs listening on this Ethernet bus.

The Security Gateway keeps the clock signal (SDC) always active so, to take control of the bus, it is necessary to "disconnect" the Security Gateway from the bus. A wire-cutting approach on the PCB was chosen for this purpose. The signals are connected to the Security Gateway during normal operation and are disconnected during switch reconfiguration.

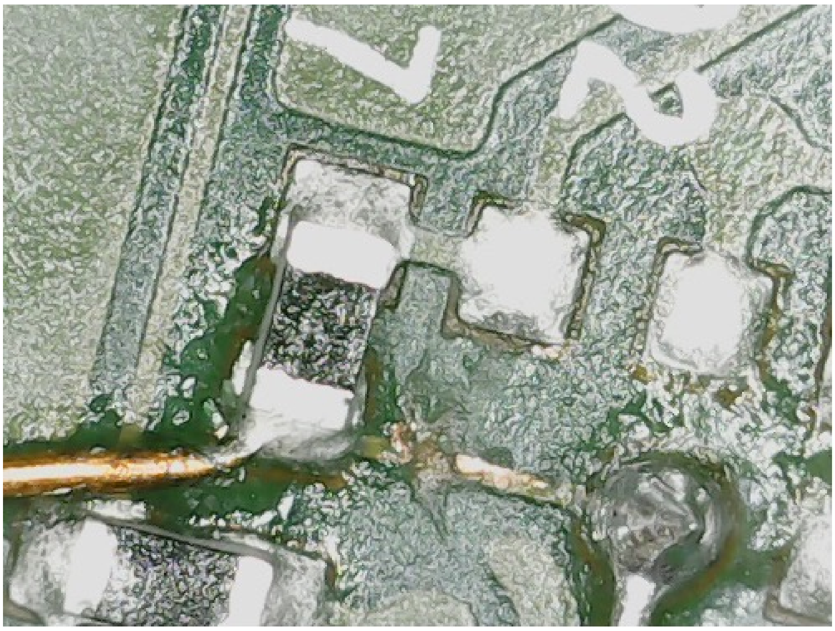


Fig. 10. Cutting the SDC signal line on the PCB

The reconfiguration is performed with a Raspberry-Pi connected to the MDIO bus, and a small Python script allowing the writing of MDIO registers to unlock the switch. The operation is as follows for each switch port:

- Disabling the port (filter configuration is only possible on a disabled port)
- Disabling filtering by modifying the PORT_PRI_OVERRIDE register
- Reactivating the port
- Placing all ports in the same VLAN

```python
mdio_bus = MDIO(clk_pin=23, data_pin=24, path='/dev/gpiochip0')
mdio_bus.open()
try:
    val=0xffff
    while val != 0x1E4F:
        val = mdio_bus.read_c22_register(0x14, 0)
        print(hex(val))
    for i in range(7):
        change_tcam_mode(mdio_bus, i)
        change_vlan(mdio_bus, i)
    val = mdio_bus.read_c22_register(0x1b, 0x1c)
    print(hex(val))
except KeyboardInterrupt:
    high_z()
```

With this reconfiguration, we were able to communicate with all components connected to the Ethernet switch from the diagnostic port.

5 Security Gateway

5.1 System

The NXP MCP5748G chip serves as the Security Gateway. This system is interconnected on one side to Ethernet and on the other side to various CAN buses.

It has two main functions:

— In normal operation, it acts as a proxy between the Ethernet and CAN worlds and performs filtering of messages it relays.

— It deploys updates to other ECUs connected to the CAN.

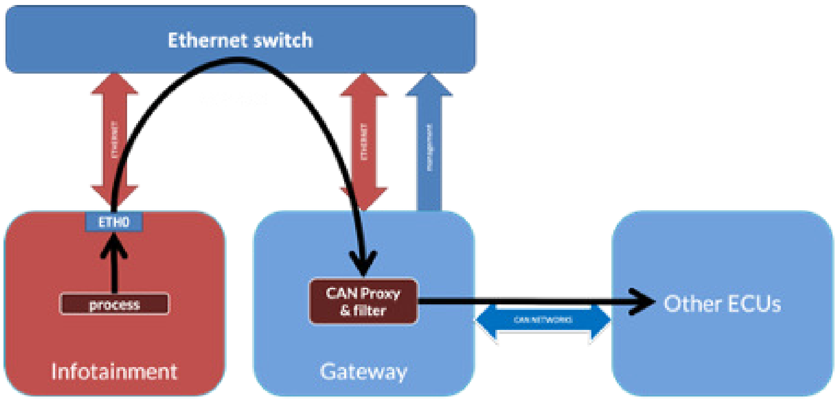


Fig. 11. Gateway architecture

The architecture is based on PowerPC-VLE and the operating system is built on FreeRTOS. Multiple software tasks provide various functionalities.

In addition to providing CAN access to the Ethernet world, this component guarantees certain vehicle information:

— The VIN (Vehicle Identification Number)

— The serial number

— The date of commissioning

— The country of marketing

— The car color

— The current security version (for anti-rollback)
— Other parameters describing the components

These data are stored in the component's internal flash memory in two memory areas, the integrity of which is verified at each startup.

An API on UDP allows reading or writing this information from the Ethernet. Writing sensitive information (such as the VIN) is subject to packet signature verification. To modify them, write requests must be signed (ECDSA). This mechanism is likely used in the factory to program these values.

5.2 Firmware

Security Gateway firmware deployment is ensured by the Infotainment, so firmware update files are available in the rootfs.

The format of the update files is simple and not encrypted so the firmware code can be extracted easily.

The PowerPC-VLE architecture is well supported by analysis tools like IDA and Ghidra, making their analysis relatively simple. The use of FreeRTOS APIs by the firmware also helps to navigate the binary for reverse engineering.

5.3 ECU update mechanism

The Security Gateway ensures the updating process of most vehicle ECUs. The update files are stored on the Infotainment system, and during the update process, the Gateway performs the following actions:

— Fetch firmware update through TFTP on the Infotainment
— Use UDS over CAN to check the prerequisites (i.e. version)
— Update the ECU with UDS messages

5.4 Secure boot

The MCP5748G does not have a secure boot mechanism. To limit the security impact of a lack of secure boot, Tesla has implemented the mechanism in the bootloader so the firmware (in internal flash or from another source) is verified at each startup.

The verification is based on an ECDSA signature for production vehicles and a simple CRC for factory mode vehicles (before provisioning).

The bootloader allows loading firmware from various media:

— SD Card
— TFTP

— Internal Flash

In the case of TFTP and the SD card, the firmware is copied to RAM, verified, and then started.

The bootloader also implements an anti-rollback mechanism. The firmware version is checked at startup and during updates. The current version is saved in a dedicated area of the internal flash memory (see System). Lower versions are not allowed to start. When an update with a higher version starts for the first time, the current value is updated.

5.5 CAN filtering

The gateway provides an UDP API on its Ethernet interface to allow other isolated components, like the Infotainment, to send CAN messages. CAN filtering by the Security Gateway is provided by whitelists; only certain CAN IDs are allowed to pass through the Gateway.

The infotainment system shares its state througt CAN messages over Ethernet. Some of these packets are interpreted by the Security Gateway and not relayed on the physical CAN bus, this is used to enable/disable certain gateway features.

6 Tesla strategy regarding external security researchers

6.1 Product security team

For a product manufacturer, dealing with the security of connected devices can be quite difficult. Indeed, it involves the security of both the company IT but also the security of the sold product.

The team at Tesla is also tasked to enhance and maintain vehicle security. They also play a crucial role in reducing the attack surface and adding new mitigations. For instance, the Tesla Blue Team recently patched Connman to disable attack surfaces that could not be disabled through compilation options alone. They also improved Connman's security by isolating the code that requires raw sockets.

Tesla have been part of the target for multiple Pwn2Own edition, they are also part of the sponsors of the event. At least 3 successful and one unsuccessful attacks on the Tesla have been demontrated during the competition (2019, 2022, 2023). Theses attacks give Tesla the opportunity to fix vulnerabilities but also to see real attack and exploit methods. Looking at attacker paths is very useful to improve the whole system to be more resistant to such exploitation techniques (hardening, design choices, sandboxing).

Alongside its Pwn2Own presence, Tesla runs a bug bounty program on the platform BugCrowd for both its infrastructure and vehicles.

6.2 Product security program

Security researchers can register their vehicles with Tesla's product security team. If a security researcher encounters software issues during testing on a registered vehicle, Tesla provides assistance to reflash the software in a service center.

If a security researcher demonstrates root access on the infotainment system, Tesla provides them with an SSH key that grants them full administrator access to their vehicle for a period of one year. This allows the security researcher to continue their research and further analyze the vehicle's security. It is an incentive that Tesla provides to security researchers to encourage collaboration and improve the security of their products.

7 Conclusion

Tesla is actively working on the security of its vehicle components from the hardware design phase to the production phase with OTA updates containing security enhancements and fixes. Numerous barriers have been put in place to reduce the risk of external attacks, limit the impact of compromises and make harder to pivot between different systems.

This makes it an interesting system to study for a security researcher as attacking it represents a fascinating technical challenge. It is hoped that other automotive manufacturers and ECUs suppliers will conduct similar work in the coming years to make this level of security the norm.

Zero Day Initiative, the company organizing the Pwn2Own competition, has announced an new edition of the competition dedicated to the automotive industry for 2024. This kind of initiative will allow participating manufacturers to measure their level of security against real attacks.

References

1. David Bérard and Vincent Dehors. Slides of the talk at Hexacon about Connman exploit. `https://www.synacktiv.com/sites/default/files/2022-10/tesla_hexacon.pdf`, 2022.
2. Inc. Ralf-Philipp Weinmann of Kunnamon and Benedikt Schmotzle of Comsecuris GmbH. T-Bone technical report. `https://kunnamon.io/tbone/tbone-v1.0-redacted.pdf`, 2019.

3. Ken Tindell and Ian Tabor. CAN Injection: keyless car theft. `https://kentindell.github.io/2023/04/03/can-injection/`, 2023.
4. Chris Valasek and Charlie Miller. Adventures in Automotive Networks and Control Units. `https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf`, 2014.