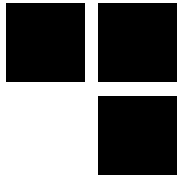


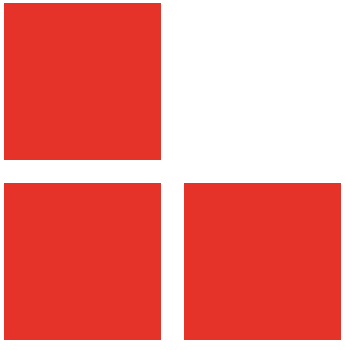
V2G Injector

Whispering to cars and charging units through the Power-Line

By Sébastien Dudek

t2.fi

October 24th 2019



Working team on the subject

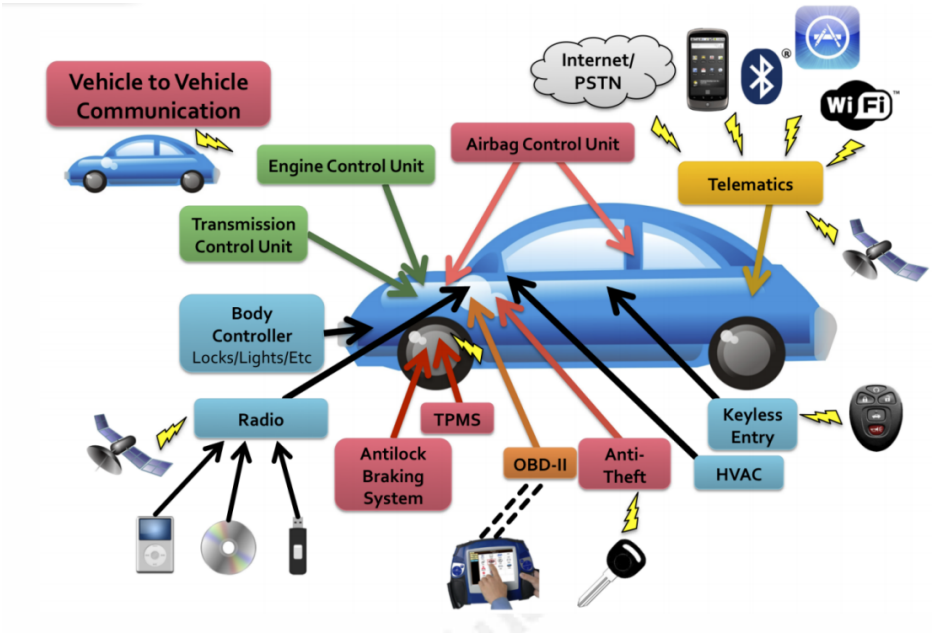
@Fist0urs, @Karion_, and me

About me

- Sébastien Dudek (@FlUxIuS)
- Working at Synacktiv pentests, red team, audits, vuln researches
- Likes radio and hardware
- And to confront theory vs. practice

Introduction

- Current cars → Controller Area Network (CAN) bus
- Engine Control Units (ECUs) → targeted via On-Board Diagnostics (OBD) port
- And plenty other surfaces to investigate:
  - Wi-Fi
  - GPRS, 3G and 4G*
  - etc.



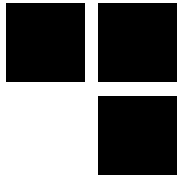
source: thetruthaboutcars.com

*https://www.synacktiv.com/ressources/Troopers_NGI_2019-Modmobtools_and_tricks.pdf

The CAN bus

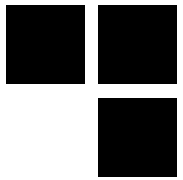
- Connecting to ODB/ODB2 interface
- Possible to interact in the CAN bus
- But too many messages are broadcasted in it $\rightarrow$ needs processing to focus on interesting messages

The CAN bus (2)



And also apparently...

Connected cars: our feedback

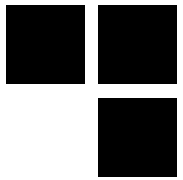


As presented at Troopers this year:

- Mobile network is generally used
- Possible to install applications
- GPRS is generally used for middle-class cars $\rightarrow$ really easy to intercept
- But parking cars are also well isolated $\rightarrow$ jamming not needed

We have also developed tools to monitor and jam 2G, 3G, and 4G cells: Modmobmap and Modmobjam.

Connected cars: beautiful features



- Enable the installation of applications
- Can be updated
- Plenty of available applications:
  - Twitter application and Facebook
  - Meteo
  - GPS
  - etc.

And all of that ”in the air”

Intercepting communications

Go further:
https://www.synacktiv.com/ressources/Troopers_NGI_2019-Modmobtools_and_tricks.pdf

Our interest: the charging connector

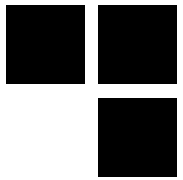
- Is it only used for charging?

Warning

Tons of abbreviations!

Let’s inspect this mysterious thing...

Long story short: renewable energy

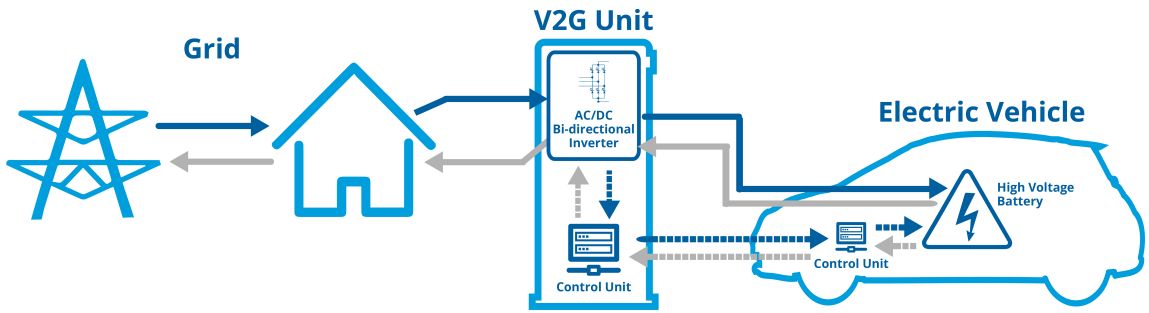


- Renewable energy production $\rightarrow$ variable and difficult to predict (solar, wind, user consumption, etc.) $\rightarrow$ Smart Grids
- People had to think about ways to store it
- First energy storage system $\rightarrow$ Battery-to-Grid (B2G)

$\rightarrow$ Why not use car's battery for energy storage too?

The rise of V2G

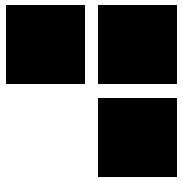
- V2G: Vehicle-to-Grid
- Use Electric Vehicles (EVs) to store energy
- In bidirectional charging/discharging systems → pay for charging or get paid → compensate battery deterioration



source: automobile-propre.com

Looking at specs → V2G systems communicate with a protocol

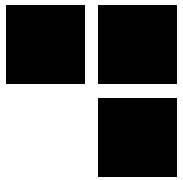
Standards for interoperability



V2G uses several standards to communicate:

- ISO/IEC 15118: Vehicle-to-Grid (V2G) communication
- IEC 61851: conductive charging system
- IEC 61850-90-8: communication networks for EVs
- and so on.

Publications

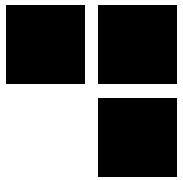


Very few of them tackle the security issues and improvements on V2G:

- Peng Wang Zhigang Ji Wenpeng Luan, Gen Li. Security of V2G Networks: A Review. Boletín Técnico, Vol.55, Issue 17, 2017
- Yan Zhang and Stein Gjessing. Securing Vehicle-to-Grid Communications in the Smart Grid. IEEE Wireless Communications, 2013.

Uses Power-Line $\rightarrow$ we published a critical vulnerability concerning DAK key generation on most HomePlug AV devices[1]

[1] http://www.nosuchcon.org/talks/2014/D1_03_Sebastien_Dudek_HomePlugAV_PLC.pdf



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

5 V2G Injector

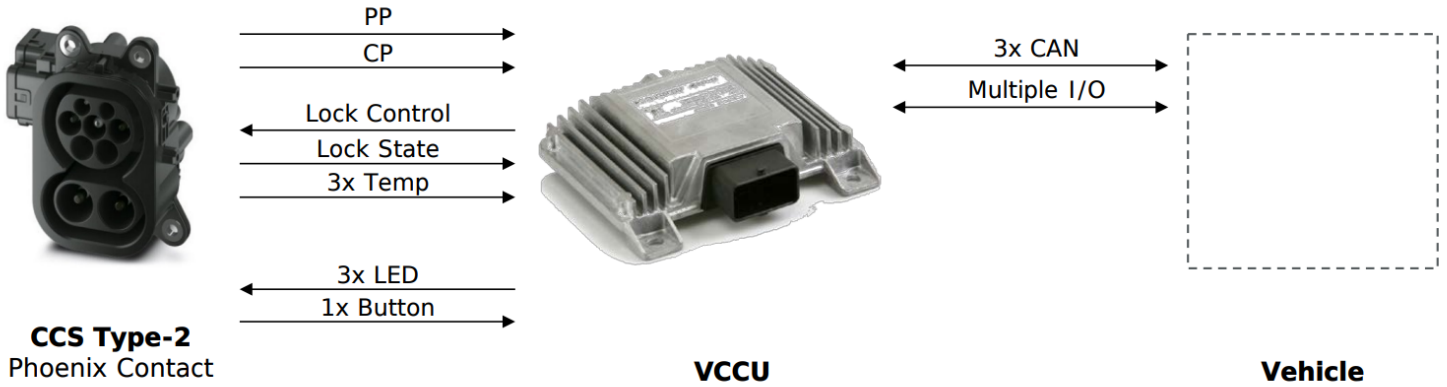
6 Attacking charging stations

7 Eavesdropping in radio

8 Conclusion

V2G ECU

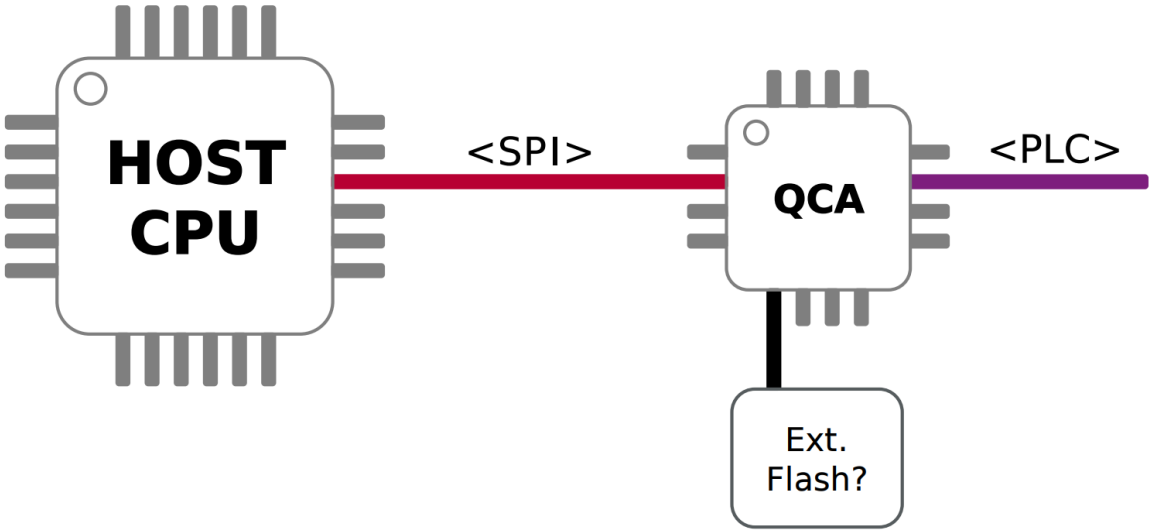
- Known as Vehicle Charging Control Unit (VCCU)
- Interfaced with a Combined Charging System (CCS)
- ECU is used for: vehicle state management, communication with the backend, coordination, etc.



source: Michael Epping. Vehicle Charging Control Unit. EMOB, 2017

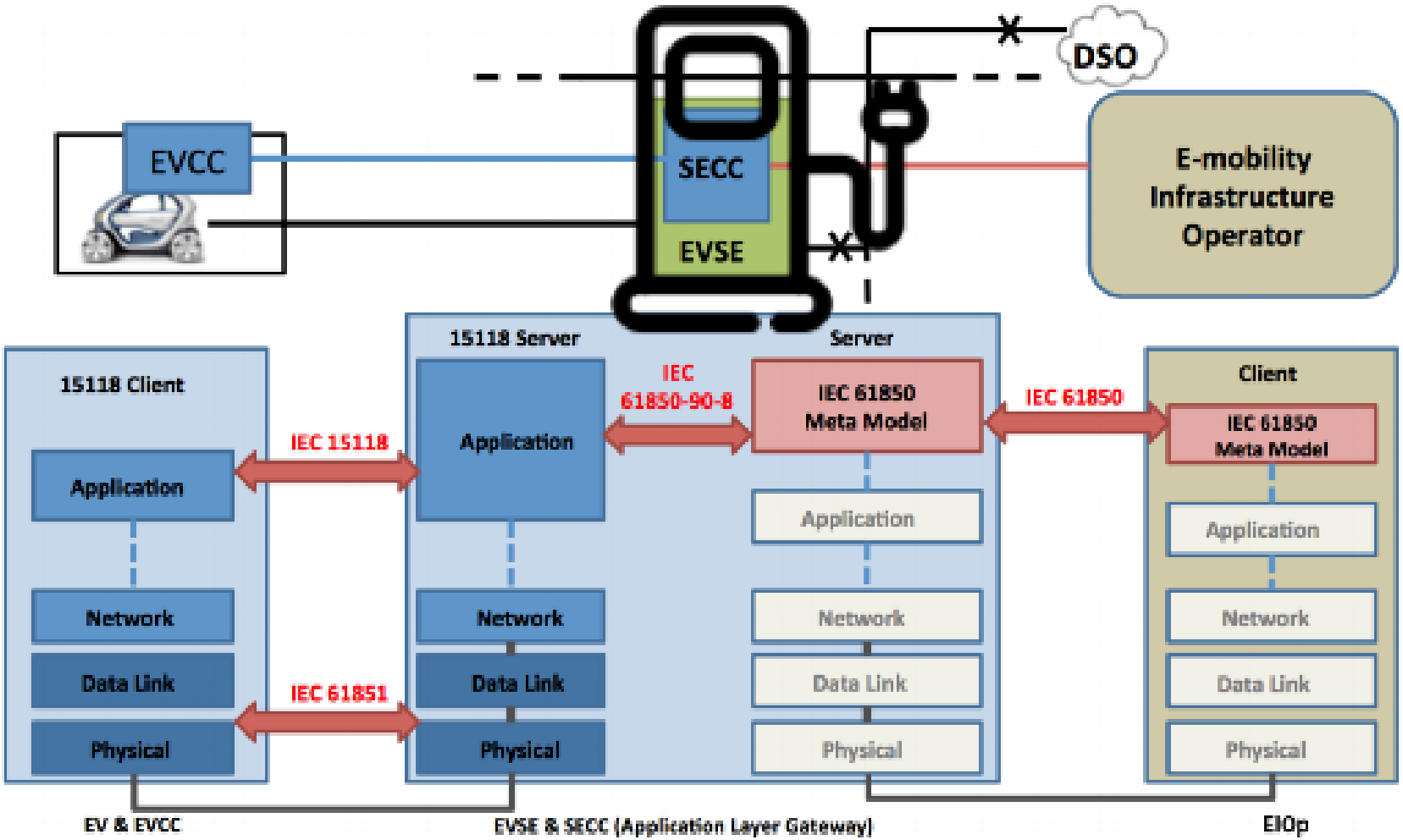
ECU parts

- 2 parts: host/app CPU and a PLC modem/baseband
- Host CPU to process data → specific to automobile area (rarely x86, ARM, MIPS, etc.)
- PLC modem to communicate in PowerLine → usually Qualcomm Atheros: QCA) → HomePlug AV/GP standard



source: Michael Epping. Vehicle Charging Control Unit. EMOB, 2017

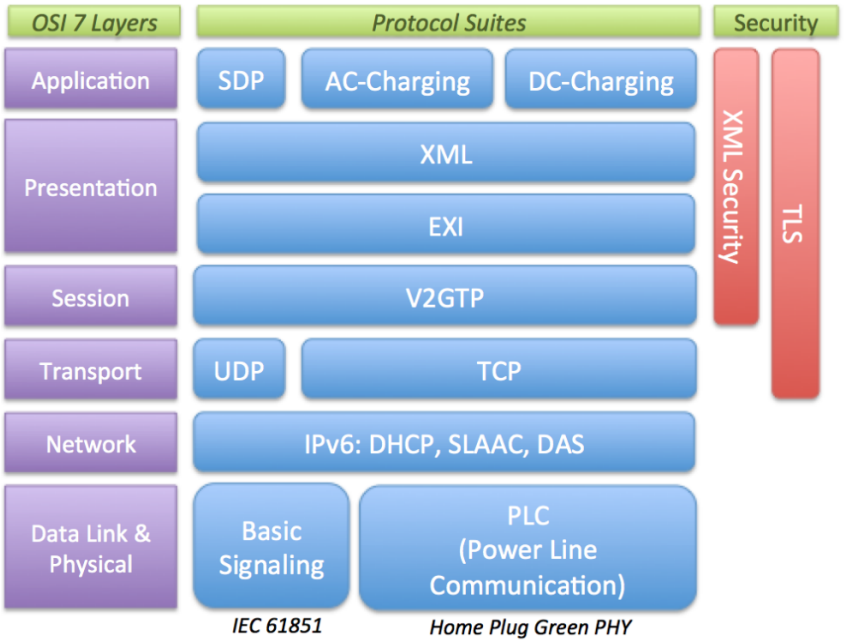
Architecture



source: https://res.mdpi.com/applsci/applsci-06- 00165/article_deploy/applsci-06-00165.pdf

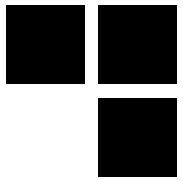
V2G layers

- L1: PHY communication via a Power-Line Communication Device
- L2: Management Message Entries (MME)
- L3: Supply Equipment Communication Controller (SECC) on $\rightarrow$ EV Supply Equipment (EVSE) host and port
- L4: V2GTP transports V2G data
- ...



source: https://res.mdpi.com/applsci/applsci-06-00165/article_deploy/applsci-06-00165.pdf

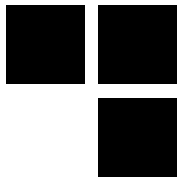
TLS with V2G data



- TLS can be enabled $\rightarrow$ usually asked by EV Communication Controller (EVCC, client part)
- Must have two distinct private keys and certificates $\rightarrow$ ensure encryption and authenticity
- Needs a Certificate Authority (CA) to check Supply Equipment Communication Controller (SECC, server part)

Interesting to test to confront specs $\leftrightarrow$ targeted implementation

TLS with V2G data

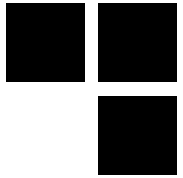


- TLS can be enabled $\rightarrow$ usually asked by EV Communication Controller (EVCC, client part)
- Must have two distinct private keys and certificates $\rightarrow$ ensure encryption and authenticity
- Needs a Certificate Authority (CA) to check Supply Equipment Communication Controller (SECC, server part)

Interesting to test to confront specs $\leftrightarrow$ targeted implementation

Reality in heterogeneous envs

Complicated to put in the chain $\rightarrow$ how vendors are dealing with it? ... ;)



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

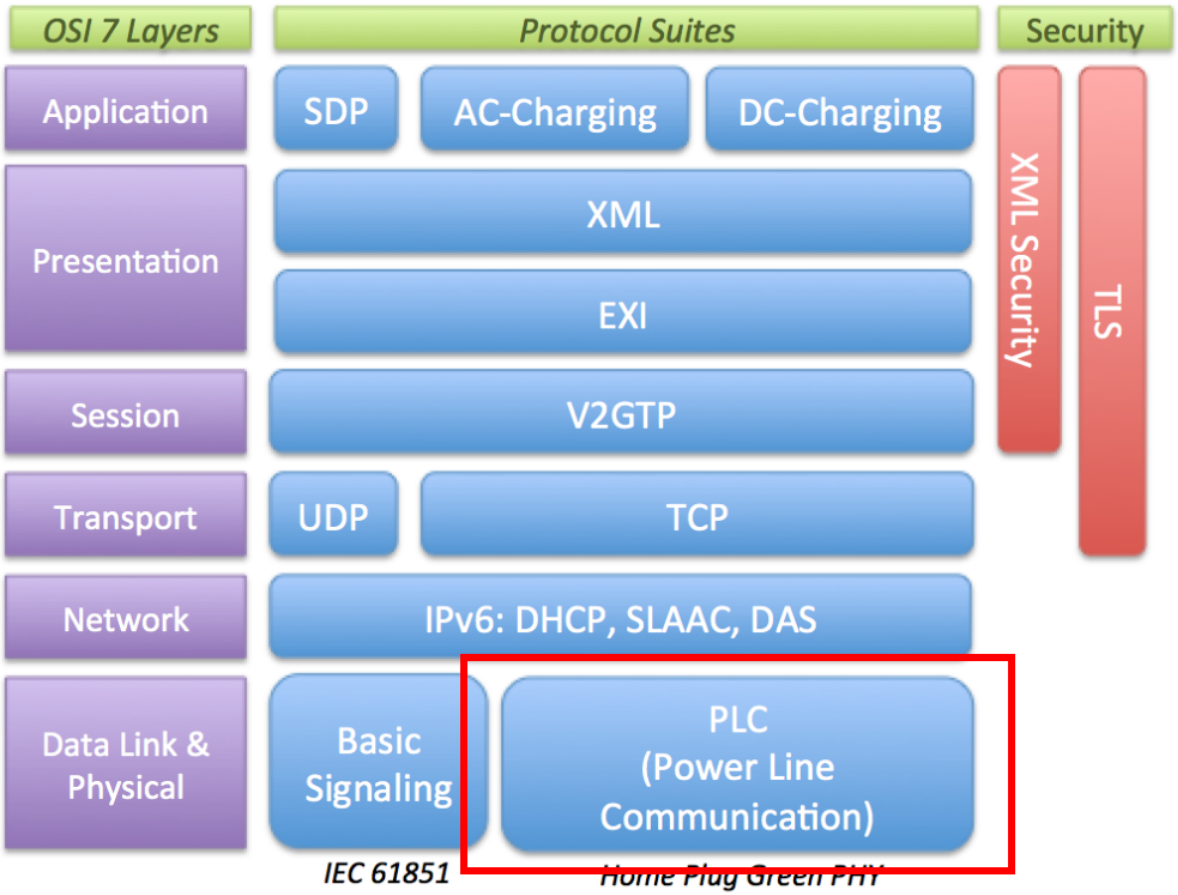
5 V2G Injector

6 Attacking charging stations

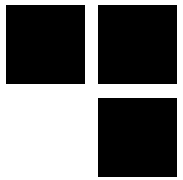
7 Eavesdropping in radio

8 Conclusion

HomePlug Green PHY



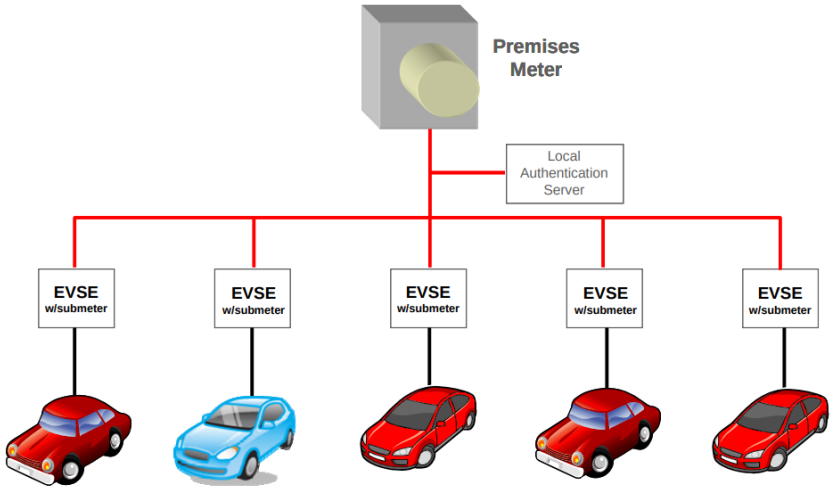
HomePlug AV and Green PHY



- HomePlug Green PHY (HPGP) $\rightarrow$ subset of HomePlug AV
- HomePlug AV used to extend domestic local network
- HPGP Intented to be used for ”smart” grid or other automation systems
- HomePlug AV higher peak rate than HomePlug Green PHY
- Keys:
  - Network Membership Key (NMK): to encrypt the communication using 128-bit AES CBC
  - Direct Access Key (DAK): to remotely configure the NMK of a targeted PLC device over the Power-Line interface

Plug-in Electrical Vehicle (PEV) Association

- PLC packets are broadcasted in the Power-Line
- So after plugging → PEV does not know on which station it is connected

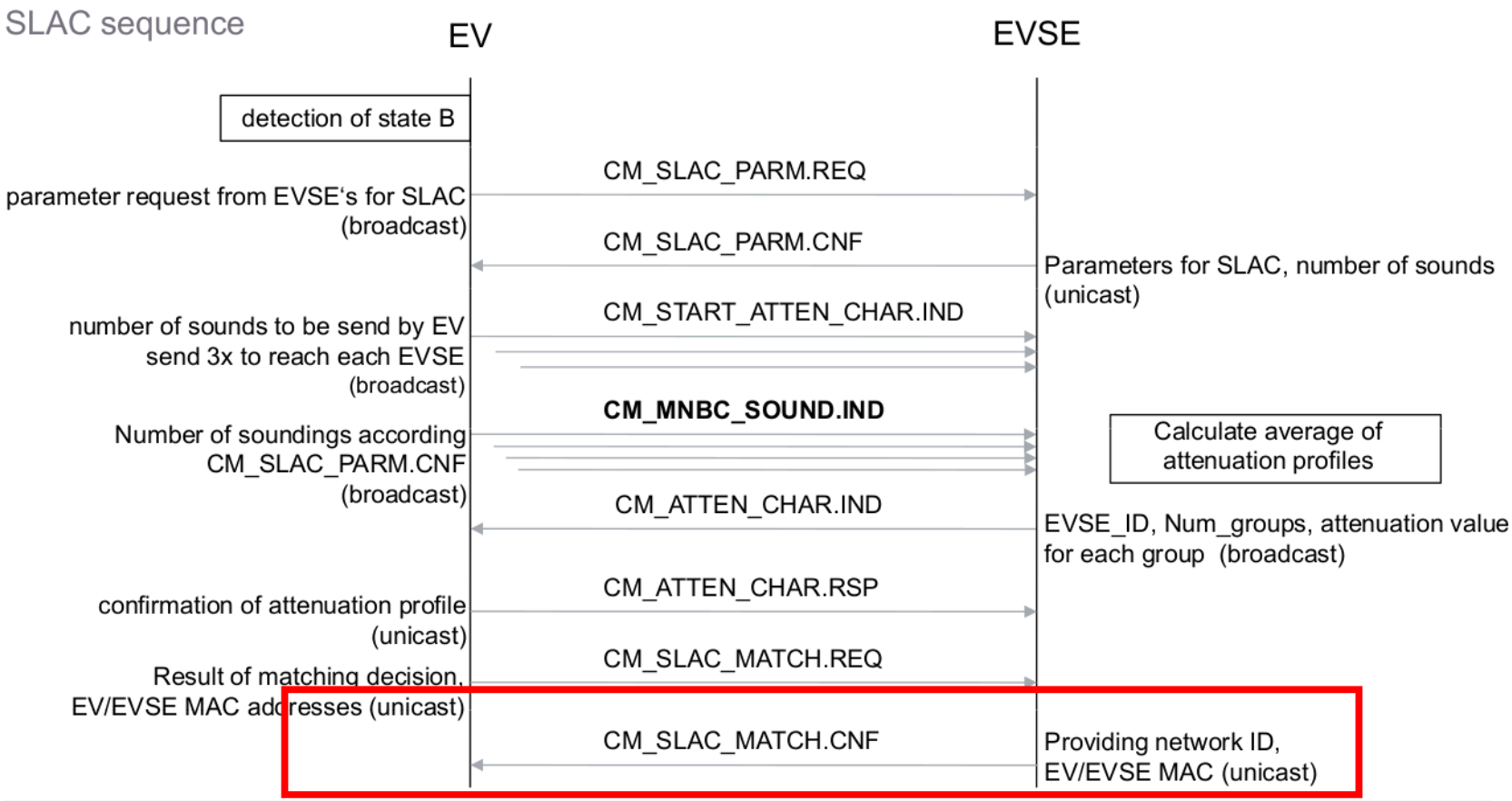


source: HomePlug Green PHY whitepaper

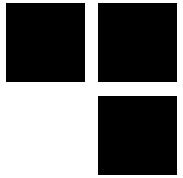
How to prevent from billing errors?

SLAC procedure

SLAC: Signal Level Attenuation Characterization



source: HomePlug Green PHY whitepaper



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

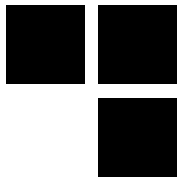
5 V2G Injector

6 Attacking charging stations

7 Eavesdropping in radio

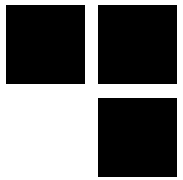
8 Conclusion

Tools and specifications



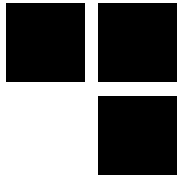
- No free specifications
- Some monitoring tools like “V2G Viewer pro” exist, but expensive
- Free and useful stacks to understand V2G:
  - RISE-V2G
  - Open V2G
- Even HPGP dissectors are publicly missing for Wireshark, Scapy, etc.

Our contribution



- Made SECC, V2GTP and HomePlug GP Scapy layers
- Developed a V2G data encoder/decoder, based on RISE-V2G shared library
- Found a new flaw in HPGP SLAC procedure
- Combined all these tools to make a tool to monitor and inject crafted packets, called “V2G Injector”

Without reinventing the wheel!



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

5 V2G Injector

6 Attacking charging stations

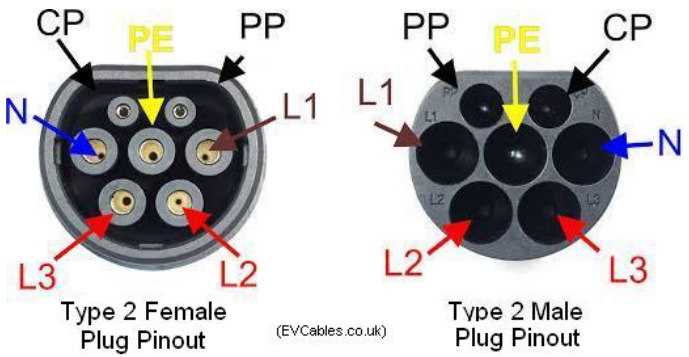
7 Eavesdropping in radio

8 Conclusion

Our interface: The Combined Charging System connectors

Different types of connectors exist, like IEC 62196 in UE:

- PP: Proximity pilot for pre-insertion signalling
- **CP: Control Pilot for post-insertion signalling**
- PE: Protective earth
- etc.



HGPG data multiplexed onto the Control Pilot and ground lines

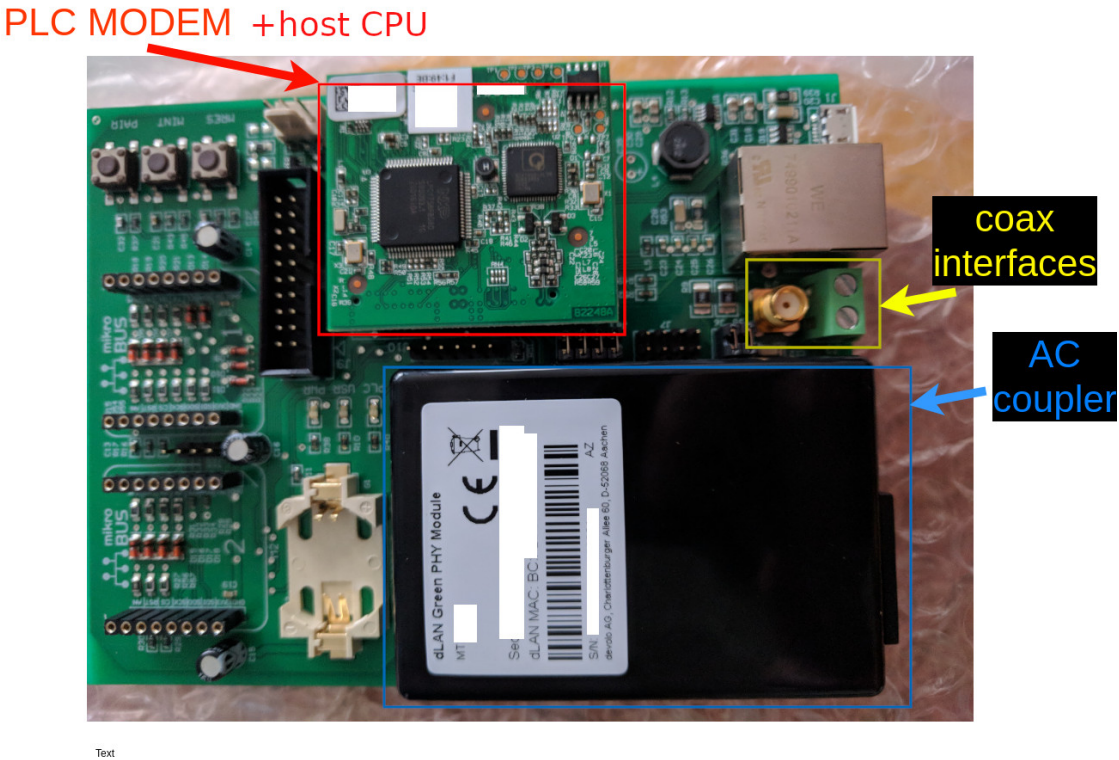
Data Propagation over Power-Line

As shown at NSC 2014 for HomePlug AV wallplugs:

- Data over Power-Line is superposed on the power supply
- Any information can propagate through many installations depending on signal strength
- If a charging station shares the same electrical network as a resident $\rightarrow$ a resident can see and contact charging station's PLC

Required hardware

- PLC with a QCA7k modem
- Tested with:
  - PLC Stamp Micro 2 Ev. Board (300€)
  - Devolo 1200+ (50€) → to rework if you want to bind it to CP lines
  - dLAN Green PHY ev. board EU II (150€):



Text

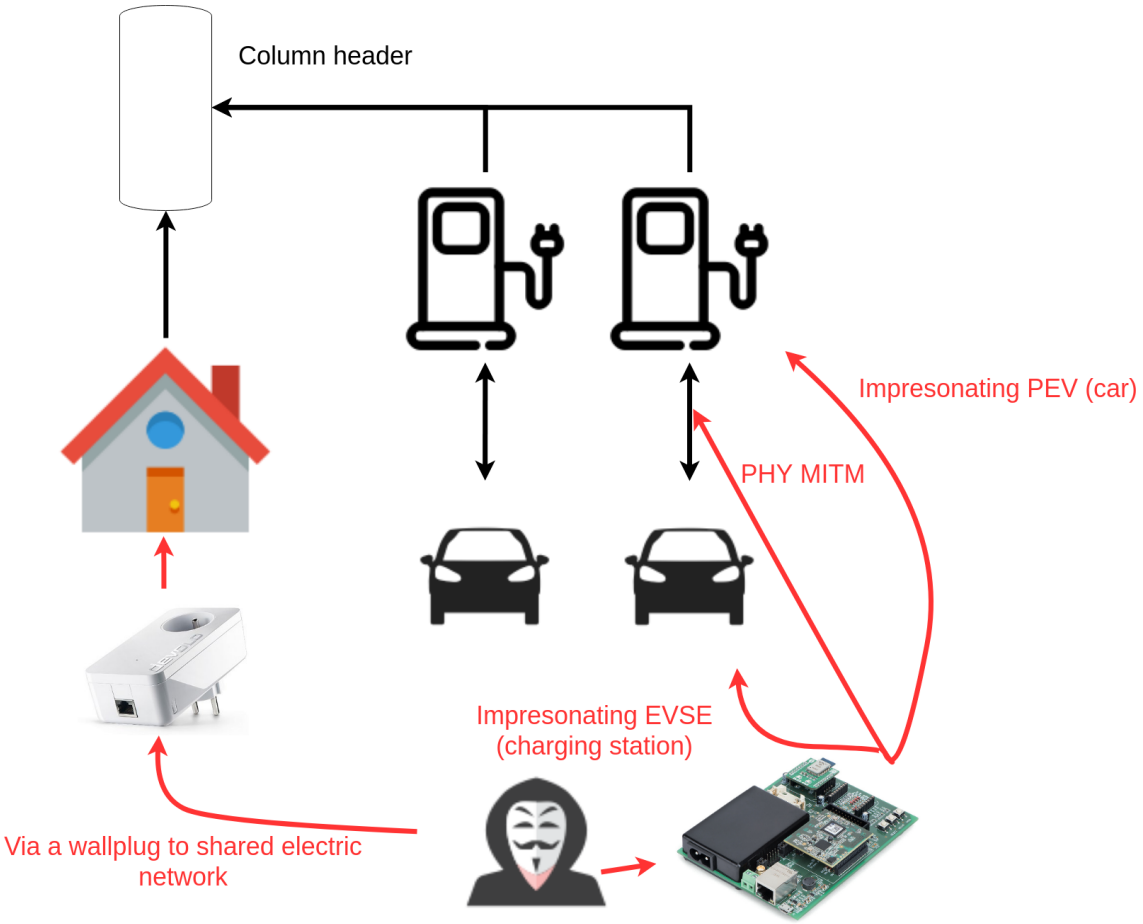
Cheapest way: the wallplug

- Devolo 1200+ works like a charm
- No modification needed if charging stations share the same electrical network
- Otherwise some rework should be done on the coupler

We had this rework in mind to document it...

And someone recently did that! :)

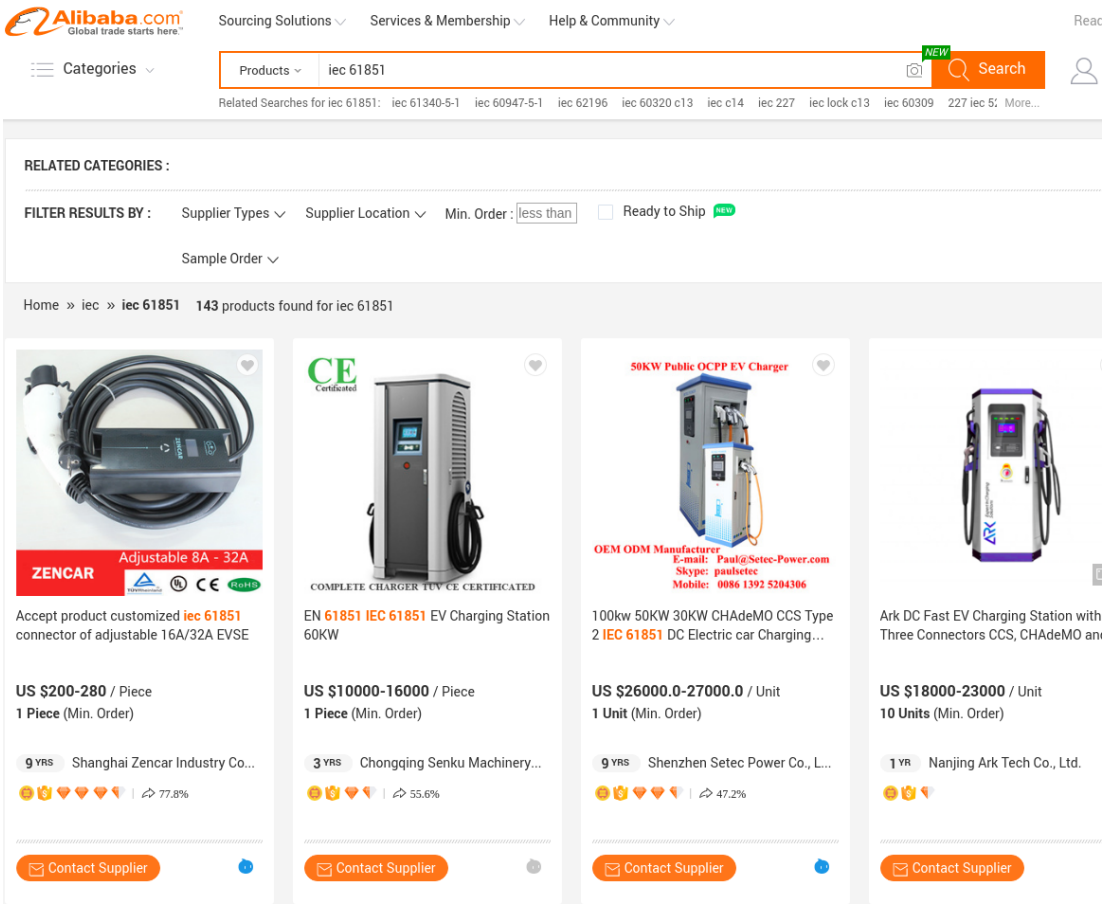
How to interface



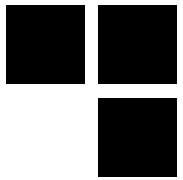
Impersonating a charging station (EVSE)

Where can we find those connectors?

You can really find everything in Alibaba, even charging stations...

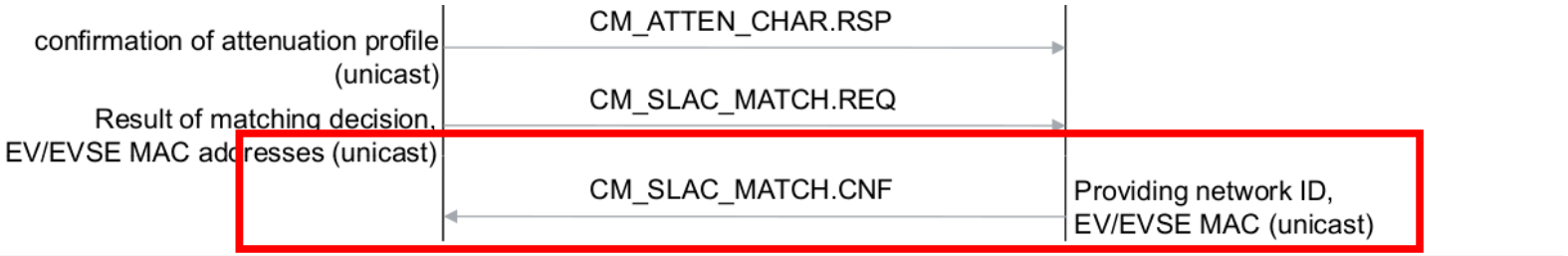


HomePlug Green PHY modes



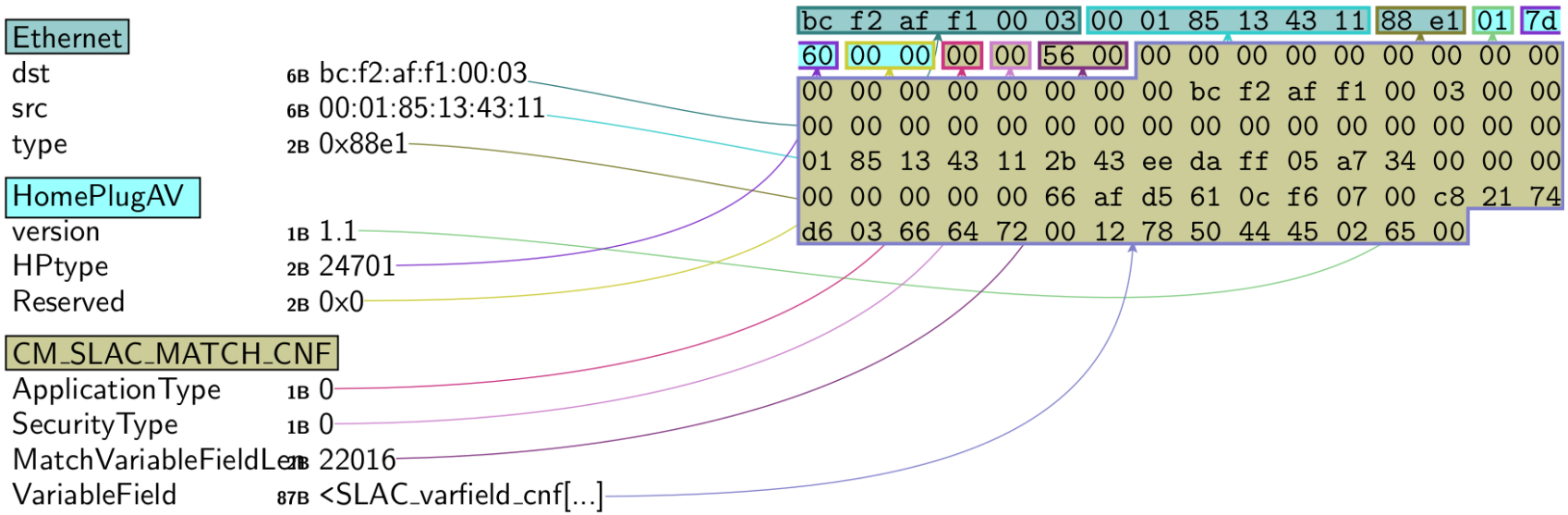
Can be set in 3 specific modes:

- Unconfigured
- EVSE (charging station): see HGPG specific packets from PEV
- **PEV (car): can see HPGP specific packets from EVSE $\rightarrow$ interesting one**



Flaw SLAC procedure

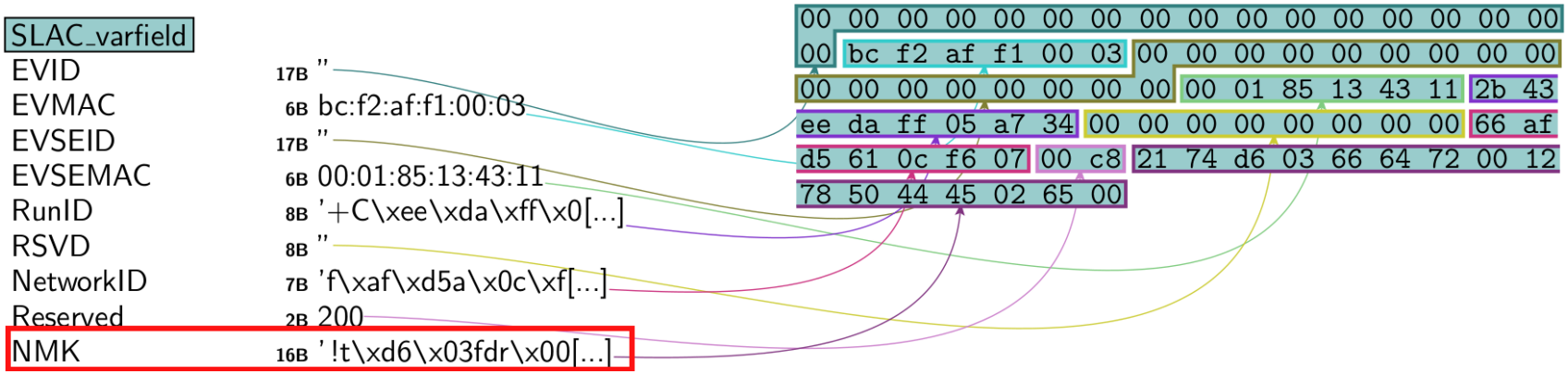
When analysing the SLAC procedure → surprise!



It was supposed to be a unicast packet, isn't it? → but it is broadcasted in the Power-Line!

Getting keys of AVLNs

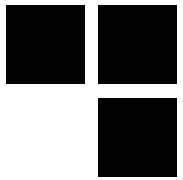
By decoding the different fields of the *CM_SLAC_MATCH.CNF* message:



Our PLC can be easily set by changing *slac/pev.ini* profile and used with *pev* tool[2]

[2] https://github.com/qca/open-plc-utils

Into the logical PLC network (AVLN)



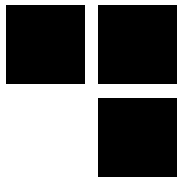
Conventional VCCU (car ECU):

1. Gets an IPv6 address
2. Looks for a V2G server $\rightarrow$ send a multicasted SECC query with required security level (encryption $\rightarrow$ *SecurityProtocol*)
3. Charging station answer giving corresponding host and port $\rightarrow$ SECC response
4. Car and charging station exchange data in V2G

Attacker

Can attack exposed services of devices and intercept communications

Intercepting communications

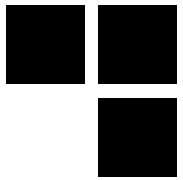


2 obvious ways:

- IPv6 neighbour spoofing attack
- Racing SECC procedure

Can also conflict MAC addresses by changing MAC address field in PLC's PIB (Program Information Block)

MITM: classic in IPv6

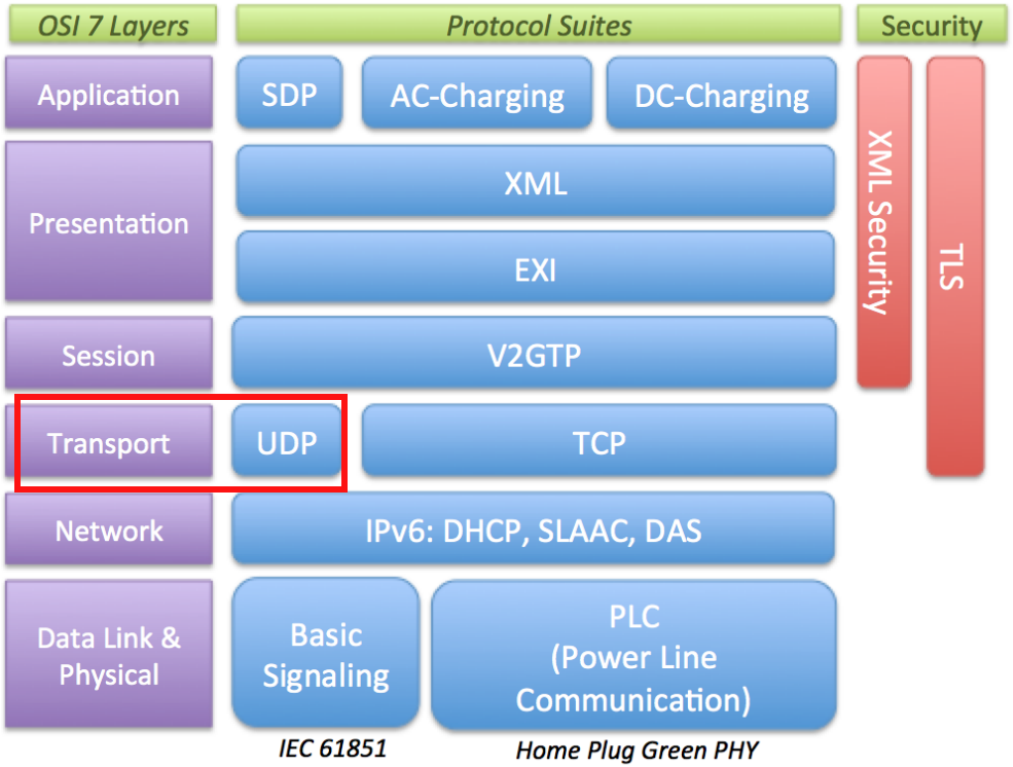


Finally, we can use the neighbour spoofing attack!:

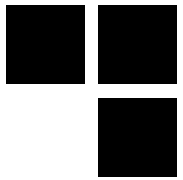
```
from scapy.all import *

while True:
    ether=(Ether(dst='bc:f2:**:**:**:**', src='00:01:87:**:**:**'))
    ipv6=IPv6(src='fe80::201:87ff:***:****',
 dst='fe80::bef2:afff:****:*', plen=RandInt(), nh=RandInt())
    hbh=IPv6ExtHdrHopByHop(options=Jumbo(jumboplen=2**30), nh=RandInt(),
len=RandInt())
    pkt = ether/ipv6/hbh
    sendp(pkt)
```

SECC procedure



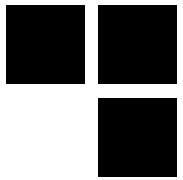
SECC procedure (2)



Clients (ECU) → SECC REQUEST in multicast:

```
###[ Ethernet ]###
[...]
###[ IPv6 ]###
[...]
###[ UDP ]###
        sport     = 60806
        dport     = 15118
        len       = 18
        chksum    = 0xc9c7
###[ SECC ]###
           Version   = 1
           Inversion = 254
           SECCType  = SECC_RequestMessage
           PayloadLen= 2
###[ SECC_RequestMessage ]###
              SecurityProtocol= 16
              TransportProtocol= 0
```

SECC procedure (3)

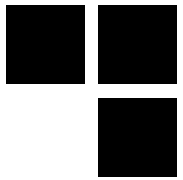


A fake station can craft an answer with fake host address and port:

```
[...]
###[ SECC ]###
           Version    = 1
           Inversion  = 254
           SECCType   = SECC_ResponseMessage
           PayloadLen= 20
###[ SECC_ResponseMessage ]###
              TargetAddress= fe80::201:85ff:fe13:4311
              TargetPort= 56330
              SecurityProtocol= 16
              TransportProtocol= 0
```

More stable than IPv6 neighbour spoofing attack

SECC procedure (3)



A fake station can craft an answer with fake host address and port:

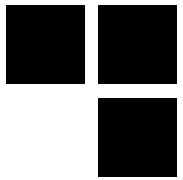
```
[...]
###[ SECC ]###
           Version    = 1
           Inversion  = 254
           SECCType   = SECC_ResponseMessage
           PayloadLen= 20
###[ SECC_ResponseMessage ]###
              TargetAddress= fe80::201:85ff:fe13:4311
              TargetPort= 56330
              SecurityProtocol= 16
              TransportProtocol= 0
```

More stable than IPv6 neighbour spoofing attack

Need to be fast

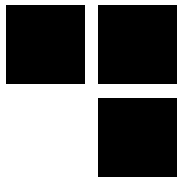
Be fast to impersonate legit SECC servers Otherwise $\rightarrow$ IPv6 neighbour spoofing

SECC: other vectors



- *SecurityProtocol* is “16” by default → for clear-text and “0” when TLS is enabled
- This field can be tricked to force the client to talk in clear-text by crafting a *SECC_ResponseMessage* with a *SecurityProtocol*=16
- Interesting to test in different implementations

SECC: attempt on a public implementation

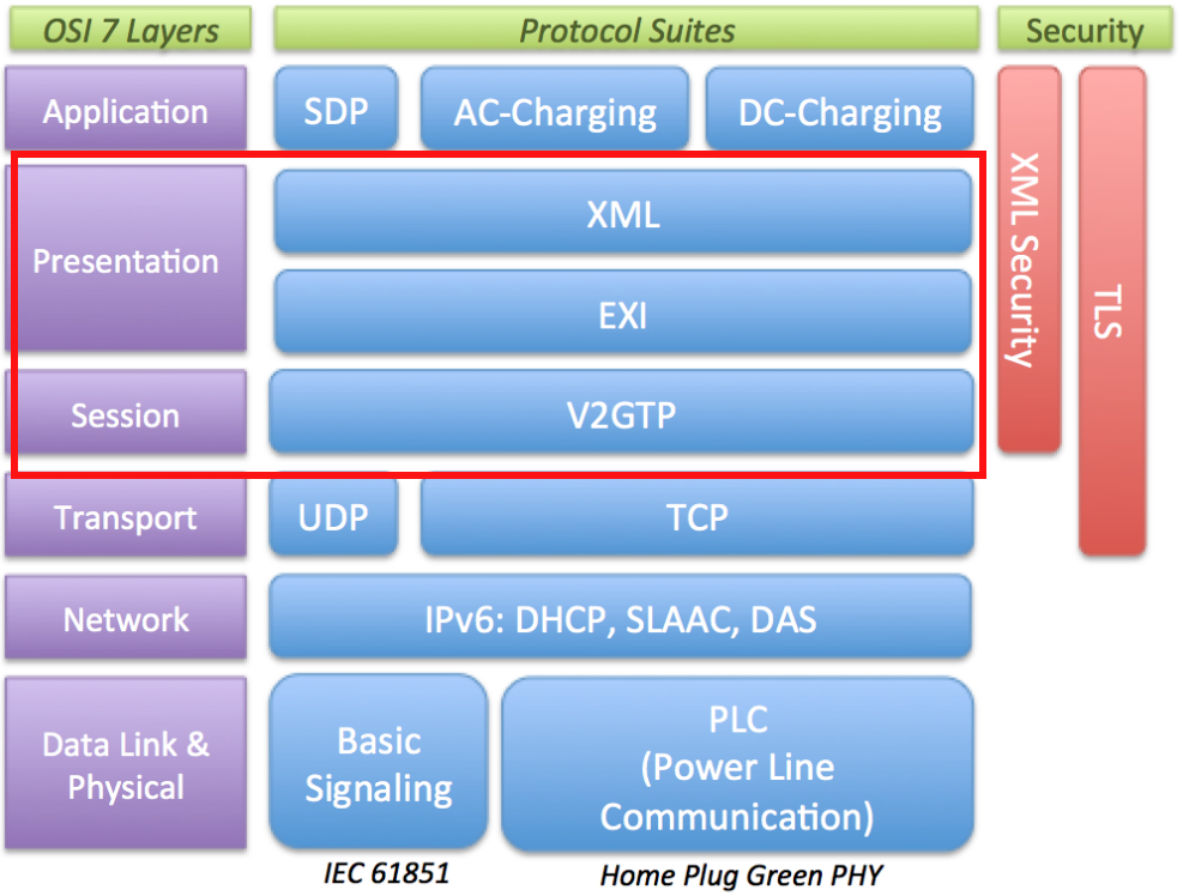


Trying this attack on RISEV2G:

```
[...]
2019-05-28T20:10:20,609 FATAL [main]
 V2GCommunicationSessionHandlerEVCC:
 EVCC and SECC could not agree on
 security level of transport layer
```

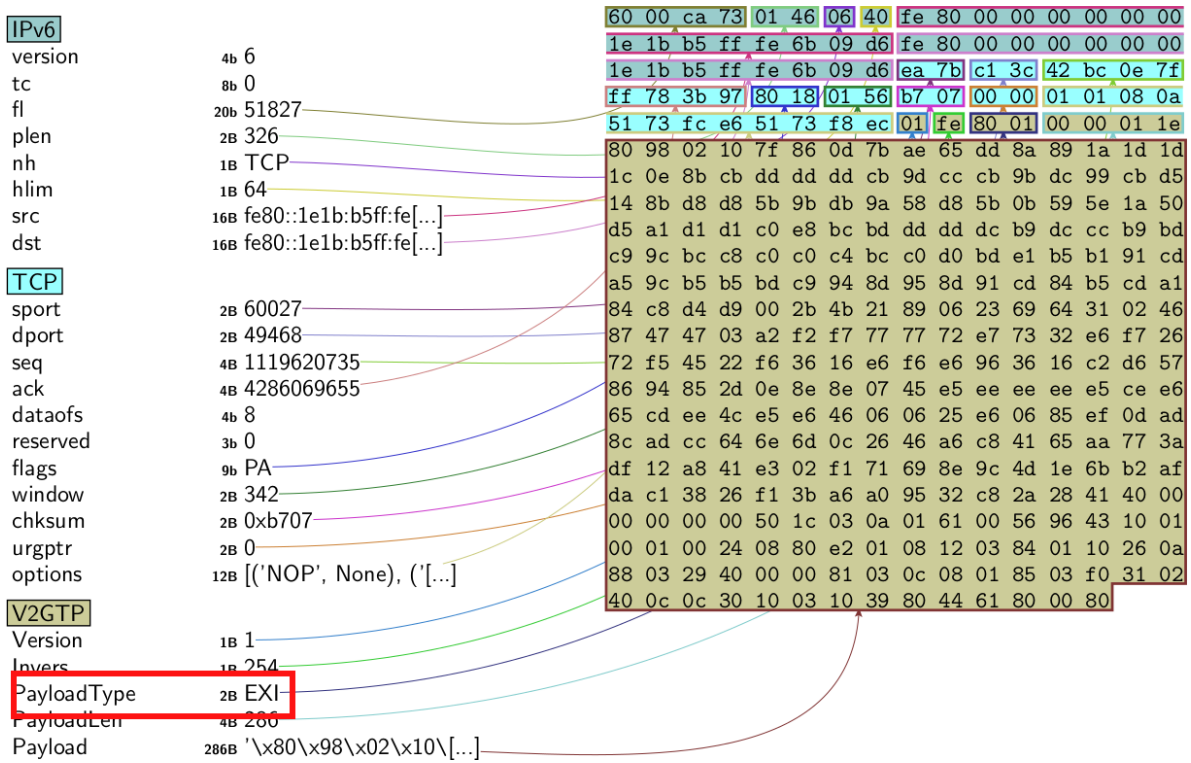
But does not mean the client is checking this level for all implementations...

V2G interception



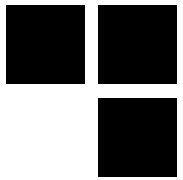
V2GTP packet

After decoding the V2GTP header:



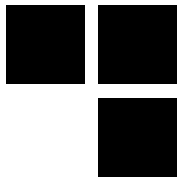
There is still unknown data in the V2GTP payload

The EXI format



- Refering IEC/ISO 15118 $\rightarrow$ data in V2G is EXI compressed
- To compress as much data $\rightarrow$ use of specific grammar $\rightarrow$ XSD schemas specific to V2G
- EXI: Efficient XML Interchange
- Aims to encode:
  - **XML (and formats using XML syntax, e.g., SVG, RSS, MathML, GraphML, ...)**
  - HTML
  - JSON
  - CSS
  - JavaScript

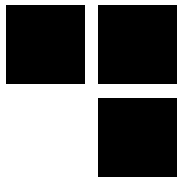
Contexts



- Each context as a XSD file, as probided in RISE V2G:
  - V2G_CI_AppProtocol.xsd
  - V2G_CI_MsgDef.xsd
  - V2G_CI_MsgHeader.xsd
  - V2G_CI_MsgBody.xsd
  - V2G_CI_MsgDataTypes.xsd
- EXI data does not provide any context

To decode EXI $\rightarrow$ RISE V2G uses state machines to select corresponding grammar $\rightarrow$ complicated in our case

Contexts



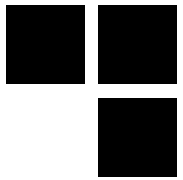
- Each context as a XSD file, as probided in RISE V2G:
  - V2G_CI_AppProtocol.xsd
  - V2G_CI_MsgDef.xsd
  - V2G_CI_MsgHeader.xsd
  - V2G_CI_MsgBody.xsd
  - V2G_CI_MsgDataTypes.xsd
- EXI data does not provide any context

To decode EXI $\rightarrow$ RISE V2G uses state machines to select corresponding grammar $\rightarrow$ complicated in our case

Circumvent: DFA

Exactly! Let's try DFA!

DFA method != Differential Fault Analysis



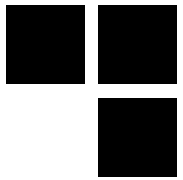
D for Dirty, **F** for fuzzy and **A** for Approach:

```
public static String fuzzyExiDecoder(String strinput, decodeMode dmode)
{
    String grammar = null;
    String result = null;

    grammar = GlobalValues.SCHEMA_PATH_MSG_BODY.toString();
    try {
        result = Exi2Xml(strinput, dmode, grammar);
    } catch (EXIException e1) {
        try {
            grammar = GlobalValues.SCHEMA_PATH_APP_PROTOCOL.toString();
            result = Exi2Xml(strinput, dmode, grammar);
        } catch (EXIException e2) {
            grammar = GlobalValues.SCHEMA_PATH_XMLDSIG.toString();
            try {
                result = Exi2Xml(strinput, dmode, grammar);
            } catch (EXIException e3) {
                // do nothing
            } catch (Exception b3) {
                b3.printStackTrace();
            }
[...]
```

in a failing order of course :)!

V2Gdecoder: decode and encode



Decode EXI:

```
$ java –jar V2Gdecoder.jar –e –s 809802107f860d7bae....
<?xml version=”1.0” encoding=”UTF–8”?><ns7:V2G_Message ...
```

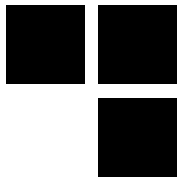
Encode XML:

```
$ java –jar V2Gdecoder.jar –x –s ’<?xml version=”1.0”
 encoding=”UTF–8”?><ns4:supportedAppProtocolReq

8000DBAB9371D3234B71D1B981899189D191818991D26B...
```

Available: https://github.com/FlUxIuS/V2Gdecoder

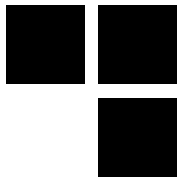
Issues with old protocols



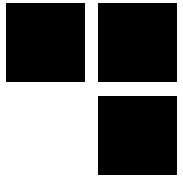
- We are able to decode first V2G packet from the car
- Contains supported application protocols including *urn:iso:15118:2:2010* → not supported in RISE V2G OSS stack → remove the XML node during a MITM

```
<?xml version="1.0" encoding="UTF-8"?>
<ns4:supportedAppProtocolReq xmlns:ns4="urn:iso:15118:2:2010:AppProtocol" ...>
    <AppProtocol>
        <ProtocolNamespace>urn:din:70121:2012:MsgDef</ProtocolNamespace>
        <VersionNumberMajor>2</VersionNumberMajor>
        <VersionNumberMinor>0</VersionNumberMinor>
        <SchemaID>0</SchemaID>
        <Priority>1</Priority>
    </AppProtocol>
    <AppProtocol>
        <ProtocolNamespace>urn:iso:15118:2:2013:MsgDef</ProtocolNamespace>
        <VersionNumberMajor>2</VersionNumberMajor><
        VersionNumberMinor>0</VersionNumberMinor>
        <SchemaID>1</SchemaID>
        <Priority>2</Priority>
    </AppProtocol>
</ns4:supportedAppProtocolReq>
```

Support for DIN 70121



- We have adapted schemas
- Based on C++ implementation in OpenV2G
- Available: https://github.com/FlUxIuS/V2Gdecoder/tree/-master/schemas_din



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

5 V2G Injector

6 Attacking charging stations

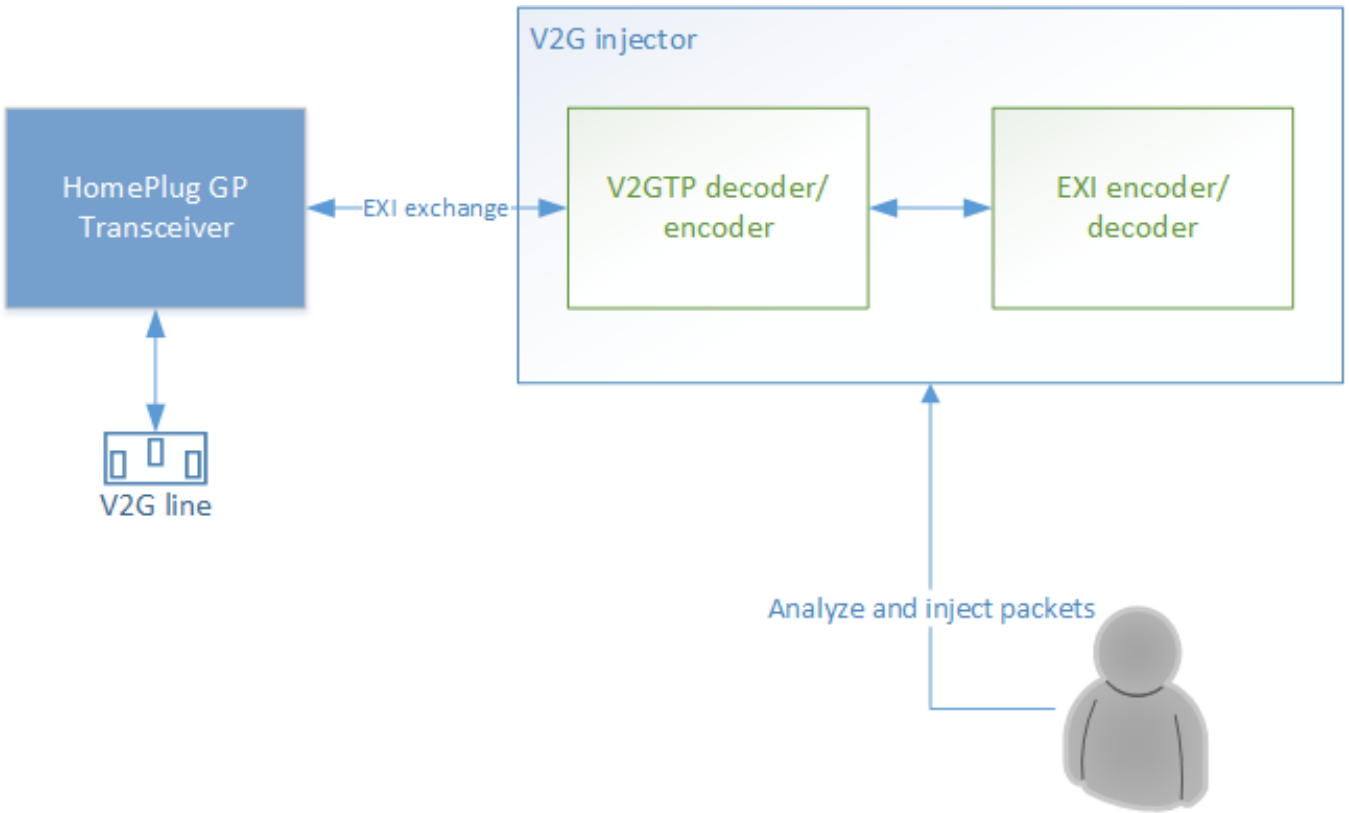
7 Eavesdropping in radio

8 Conclusion

Rise of the HPGPhoenix

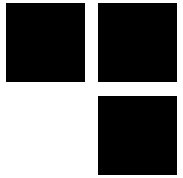
Available: https://github.com/FlUxIuS/V2GInjector

Simple architecture



We can intrude a HPGP network, and analyse/decode/encode/inject V2G data

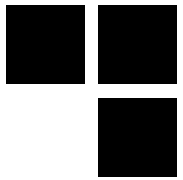
HPGP keys



Automatically done:

```
~>>> n=Network()
~>>> n.sniff(iface="eth0")
[...]
[New HPGP network spotted!]
— EVSEID: '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
— NetID: '\xae\x20\x00\xff\x82\x02\x00'
— NMK: '\x43F\xc8\xaeT\xbf\xefs\x01\x84\x94\xf8\xc3\x17'
— EVID: '\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff'
— RunID: '\xef\x34C\xf5E\xe0\xa6\x01'
```

Generate V2G packets



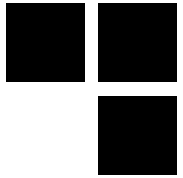
Use the dedicated Scapy layers:

```
~>>> ether = Ether()
~>>> ip = IPv6(dst="fe80::3e2a:b4ff:3e5f:1a4")
~>>> tcp = TCP(sport=6666, dport=54054, flags=24)
~>>> v2g=V2GTP()
~>>> packet = ether/ip/tcp/v2g
~>>> packet
<Ether  type=0x86dd |<IPv6  nh=TCP dst=fe80::3e2a:b4ff:3e5f:1a4 |
<TCP  sport=6666 dport=54054 flags=PA |<V2GTP  |>>>>
```

XML → compressed in EXI → included in the V2GTP payload:

```
~>>> xml = '<?xml version="1.0" encoding="UTF-8"?><ns7:V2G_Message ....
</ns7:V2G_Message>'
~>>> encoded_xml=encodeEXI(xml)
~>>> encoded_xml
u'8098020000000000000011D018706ED5AC275800'
~>>> packet.Payload=encoded_xml
~>>> packet
<Ether  type=0x86dd |<IPv6  nh=TCP dst=fe80::3e2a:b4ff:3e5f:1a4 |
<TCP  sport=6666 dport=54054 flags=PA |
<V2GTP  Payload='8098020000000000000011D018706ED5AC275800' |>>>>
```

Then send it using *sendp()* function.



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

5 V2G Injector

6 Attacking charging stations

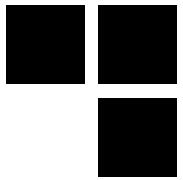
7 Eavesdropping in radio

8 Conclusion

Few words on public charging stations

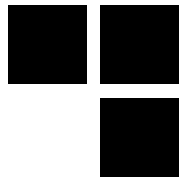
- Runs a complex OS (Linux or WinXP CE generally)
- Some available services:
  - V2G webservice
  - SSH
  - Web console/management/log interface
  - Sometimes: Telnet and more...
- Connected to an operator
- If attacked $\rightarrow$ used as pivot

Recent attacks on EVLink Parking



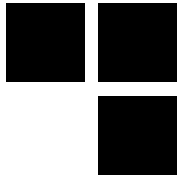
- Three vulnerabilities with a physical access by Positive Technologies:
  - CVE-2018-7800: A Hard-coded Credentials vulnerability exists which could enable an attacker to gain access to the device;
  - CVE-2018-7801: A Code Injection vulnerability exists which could enable access with maximum privileges when a remote code execution is performed;
  - CVE-2018-7802: A SQL Injection vulnerability exists which could give access to the web interface with full privileges.
- Same could be found remotely with a Power-Line Communication attack on GP!

Attacking charging stations



1. Plug the kit with the right adapter directly to the charging station;
2. Launch *slac/pev* of *open-plc-utils* with right *pev.ini* configuration profile
3. Wait until the PEV client negociates the NMK with charging station and use this key to join your AVLN
4. Start to ”nmap” and have fun! :)

To be also featured in HomePlugPWN soon!



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

5 V2G Injector

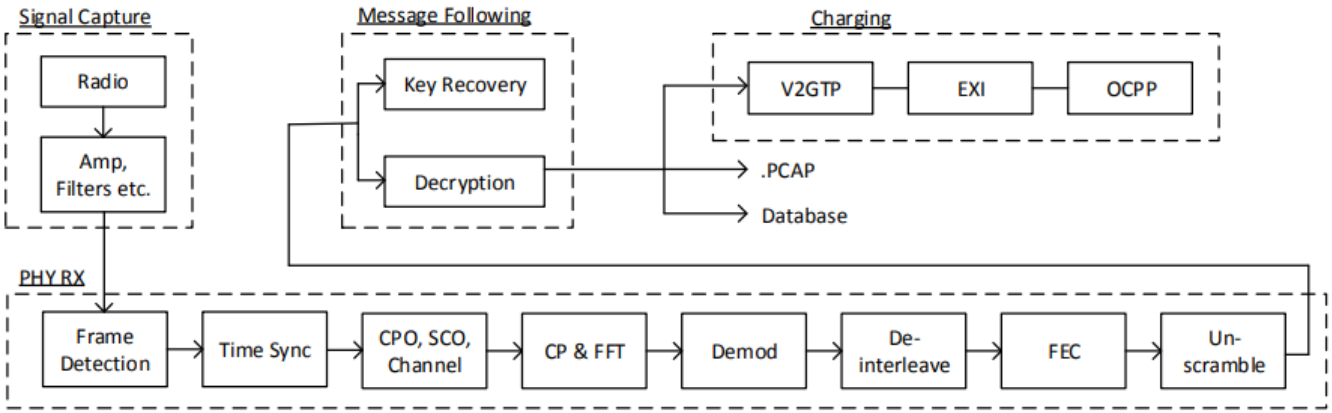
6 Attacking charging stations

7 Eavesdropping in radio

8 Conclusion

EM attack

- As well as an electrical home network → charging cable → unintentional antenna
- It could be possible to eavedrop communications
- Attack demonstrated by Richard Baker and Ivan Martinovic [3]
- Same conclusions on NMK confidentiality and V2G PKI



[3] https://www.usenix.org/system/files/sec19-baker.pdf

EM attack (2)

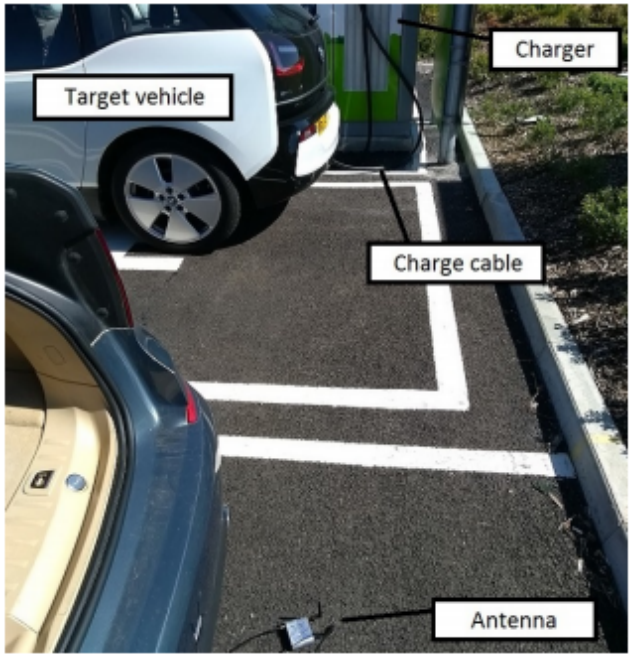


Figure 7: Eavesdropping from the next parking bay (site G), more than 4 metres away on the other side to the charging cable. In this arrangement 91.8% of messages were received successfully.

Figure 8: Two vehicles charging simultaneously. With the eavesdropper between the two vehicles 42.5% of messages were received successfully, including the NMK key establishment for both vehicles.

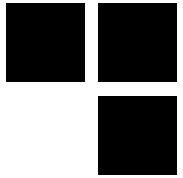
Source: https://www.usenix.org/system/files/sec19-baker.pdf

$\rightarrow$ Needs good equipment to capture all symbols + process them!

Improvements of the attack

- Use the dedicated hardware
- Direct Memory Accesses exist using the SPI interface:
  - *qcaspi_write_burst*, *qcaspi_receive*, etc.
  - Work in progress to get additional/hidden frames

Few more infos:
https://www.synacktiv.com/ressources/leHack2019-Return_of_FAIFA_and_HomePlugPWN-dudek.pdf



1 V2G communication

2 HomePlug Green PHY

3 Preliminaries

4 Intruding a V2G network

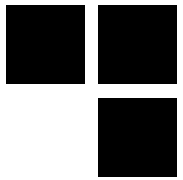
5 V2G Injector

6 Attacking charging stations

7 Eavesdropping in radio

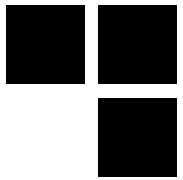
8 Conclusion

Conclusion



- V2G opens new interesting surfaces
- We have developed a tool to play with it → V2G Injector
- The project is free to use and also to contribute ;)
- ECU are less featured than charging stations
- Intruding charging station could lead to interesting pivots
- Further work:
  - DMA attacks on the dedicated hardware
  - Add a complete simulator
  - more EXI grammars
  - Add attacks and fuzzing wrappers for SECC, V2GTP, EXI and HomePlug GP

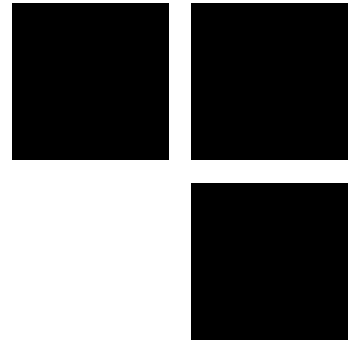
Other areas of research



- EXI format fuzzing [4]:
  - Fuzzing from XML $\rightarrow$ difficult as XML are parsed and processed against XSD
  - Better chances with the compressed data against C/C++ implementations $\rightarrow$ AFL for the road
  - Real ECUs' firmware use a proprietary EXI decoders
  - But public EXI libraries could be interesting to attack charging stations

[4] Suggested also by @agarri_fr :)

ANY QUESTIONS?



THANK YOU FOR YOUR ATTENTION,

SYNACKTIV
DIGITAL SECURITY