#RSAC

RSAConference2018

San Francisco | April 16 – 20 | Moscone Center

MATTERS NOW

SESSION ID: SBX3-W1

IS CAR HACKING OVER? AUTOSAR SECURE ONBOARD COMMUNICATION

Jeffrey Quesnelle

Director of Software Development
Intrepid Control Systems
@IntrepidControl

Introduction

- Spent 15 years working for a global tools supplier in automotive networking
- Everything we will talk about today is from open standards
- Perquisite knowledge: What is CAN bus? What is an ECU?
- Follow along with the slides: http://jeffq.com/rsa_secoc.pptx

The Insecure Era

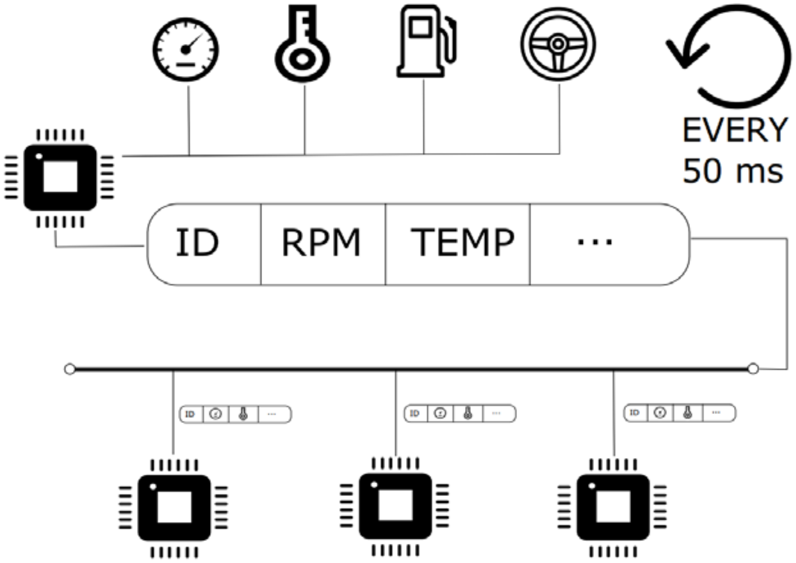
- In automotive networking we’re still in an “insecure by default” mindset
- Except for a few cases (i.e. the immobilizer) the messages are accepted “as-is”
- Helpful for when simulating and testing, but potentially dangerous in the real world
  - In general, an attacker that can send arbitrary CAN messages on a vehicle has complete power

Implicit Availability

- The dominant paradigm in automotive networking in implicit availability
- The ID of the message refers to the contents of the message
- Leads to nodes that are highly decoupled from each other
  - Low end car doesn't have backup camera? Okay, no backup camera messages
  - Swap in a new node if one goes bad
  - Complete separation of concerns

Implicit Availability

- Any node could send RPM message
  - Would be accepted as genuine
- "Okay" since vehicles were the ultimate air-gapped network
- Changing rapidly as cars become more connected
- We would like a system that prohibits this
  - But maintains advantages of implicit availability



Threat Model

- When designing a security system we need to analyze our threat model
  - What are we trying to prevent?
  - What are the likely attacks our system might face?
  - Assuming some of our defenses are breached, what are the consequences? Can a partial compromise be mitigated?
- To help us create our threat model, let's look at some recent attacks on automotive networks

Miller, C. and Valasek, C. 2015

Remote Exploitation of an Unaltered Passenger Vehicle aka Uconnect hack

- Certain 2013-2015 FCA vehicles with “Uconnect” had head units that were listening on port 6667 on 3G modem (public IP address)
  - Port 6667 was D-Bus, a Linux remote procedure call (RPC) protocol
- Could remotely issue D-Bus commands to perform any action the head unit could (control HVAC, music, etc.)
  - Head unit not directly connected to CAN bus – proxied through a microcontroller over SPI
  - D-Bus service to reflash the micro
    - Firmware file was not authenticated
- Attackers remotely reflashed the micro with custom firmware giving remote arbitrary CAN access

Thuen, C. 2015

Remote Control Automobiles aka Progressive Snapshot hack

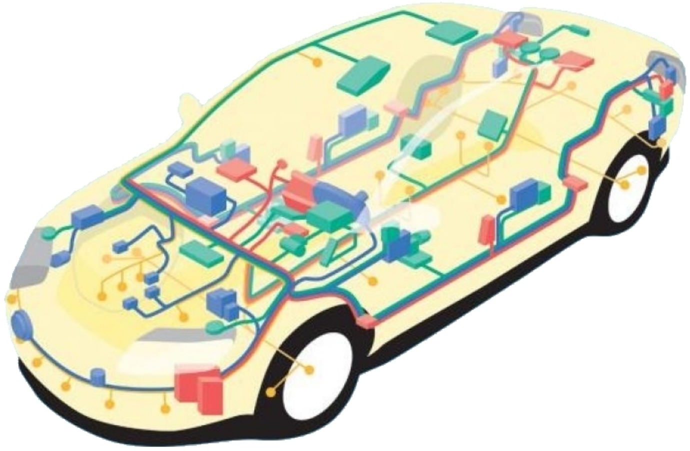
- Progressive Insurance gave customers an OBD-II dongle to plug into their vehicles
- Monitored standard OBD-II PIDs
  - Discounts are offered for “good” drivers
- Dongle contained modem which reported data back to Progressive servers
- No cellular authentication, anyone with a SDR could simulate a base station and send commands to the dongle
- As with Miller 2015 there was no authentication of firmware updates

Threat Model

- The Uconnect hack prototypical is threat – remote vulnerability on an OEM module with full network access
  - Complete compromise
- The Progressive hack can be mitigated through alternative measures
- Goal: A system that
  - Maintains the advantages of our current network architectures
  - Ensures that messages are only produced and consumed by the intended nodes

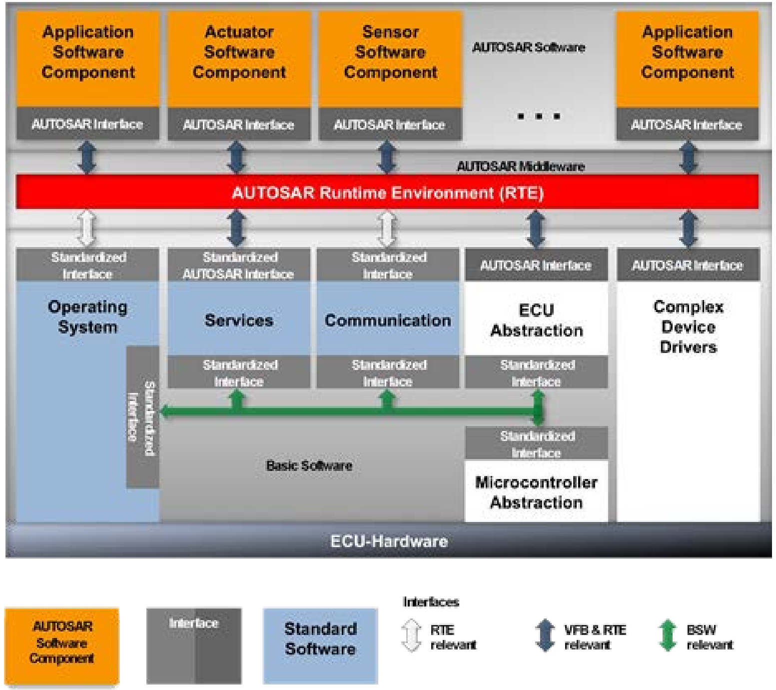
AUTOSAR

- Cars are like Lego models – little pieces all snapped together
  - One car may have ECUs developed by ten different vendors
  - Even the same ECU may have software developed by multiple parties
- AUTOSAR is a worldwide partnership of defines standards for software architecture
- Open specifications for each different software layer
- Gained widespread market adoption
  - Estimated that by 2020 every car will have AUTOSAR based ECUs
  - If you're pentesting an ECU, it probably uses AUTOSAR



AUTOSAR Secure Onboard Communications

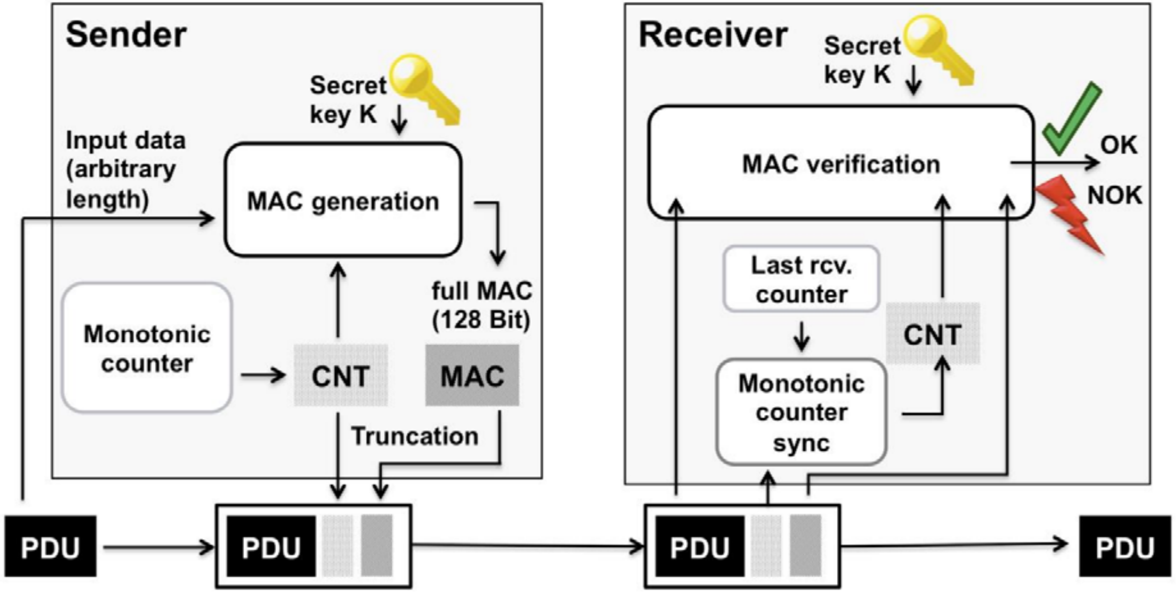
- **SecOC** is an AUTOSAR module that provides PDU (message) integrity and authentication
- Ensures the "freshness" of the PDU (protecting against replay attacks)
- Generic system that can use either symmetric or asymmetric cryptography
- Not specified: key distribution
  - Greatly affects the effectiveness of the system



Data ID and Freshness Value

- Every PDU has a unique numeric identifier known as the SecOCDataID
  - For example on CAN networks this can just be the ID of the message
- Every sender/receiver of a PDU must maintain some freshness state for that PDU
- SecOC suggests two different freshness strategies
  - Freshness timestamps
  - Freshness counters

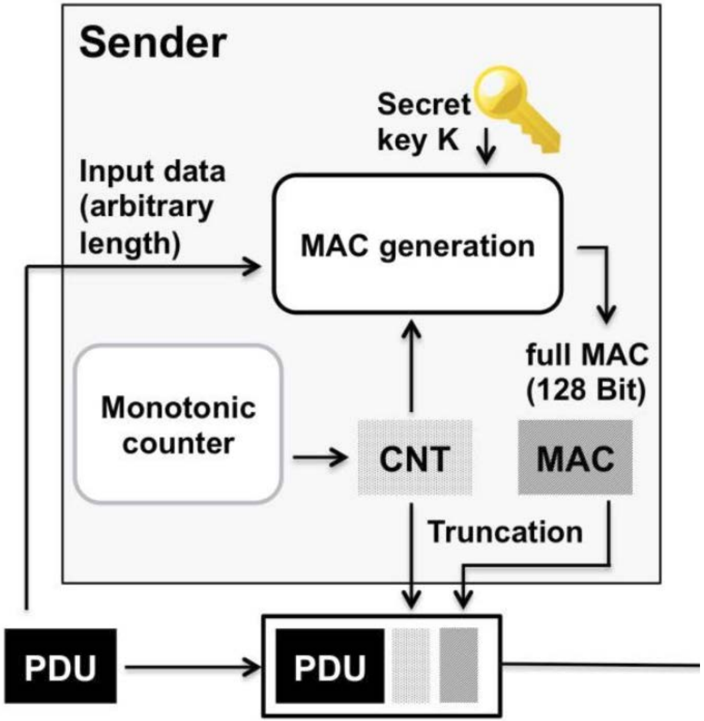
SecOC



SecOC in counters mode using symmetric crypto

Creating a Secured I-PDU

- A Secured I-PDU contains the original message (the Authentic I-PDU) as well as freshness value and the MAC
- Freshness value is incremented on every transmit
- Input to the MAC/digital signature algorithm is calculated as SecOCDataID + AuthenticIPDU + FreshnessValue
- Use secret key on the data to create the authenticator
- In symmetric mode simply chop off some MAC bits
  - Security decreases linearly with MAC size



Creating a Secured I-PDU

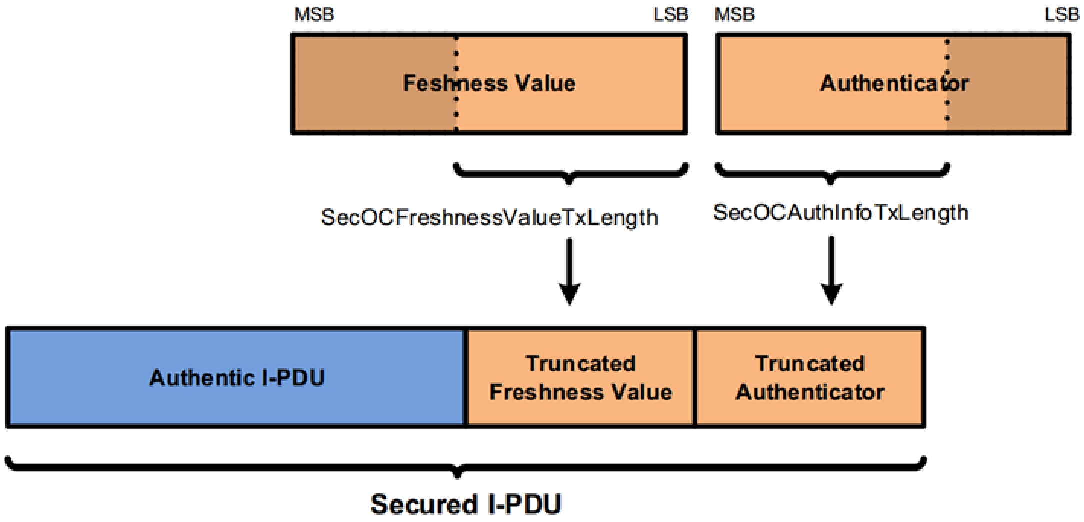
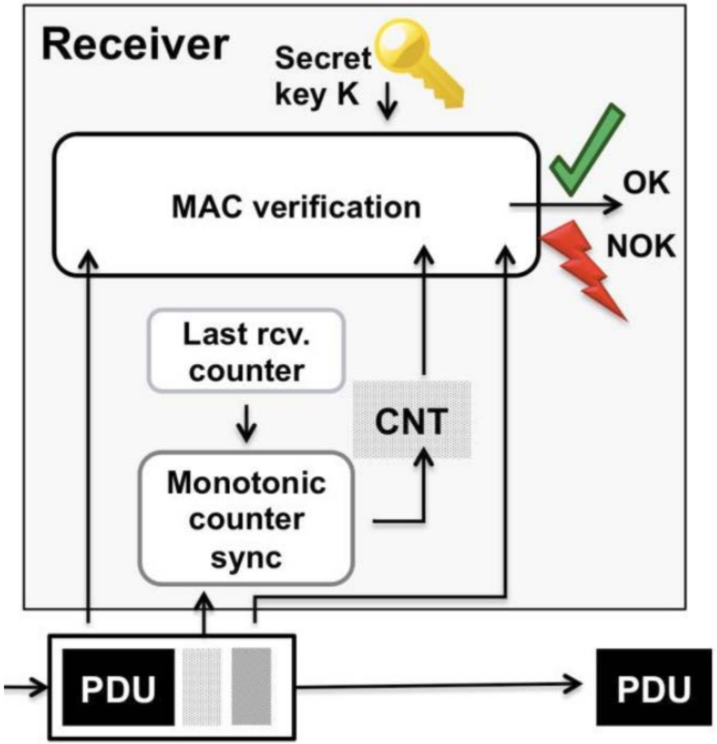


Diagram of a symmetric Secured I-PDU

Verifying a Secured I-PDU

- Only the least significant bits (LSBs) of the freshness value are sent
- Compute full candidate freshness value by overwriting the LSBs of the last received value
  - If the LSBs from the incoming I-PDU are less than those of the last received value we increment the MSBs
    - Forces the counter or timestamp to always be monotonically increasing
- Calculate the authenticator value for the received Authentic I-PDU using secret key, ID, full candidate freshness value, and the date bytes
- If the authenticator value matches, accept the I-PDU and set the new freshness value
  - Otherwise the I-PDU is rejected



Analysis of SecOC

- SecOC uses cryptographic primitives to ensure the integrity and authenticity of messages
- Maintains most advantages of implicit availability
- The freshness value is independently maintained by the sender and receiver and changes on each transmission
  - Since this value is included in the data sent to the authentication algorithm each authenticator will be different
  - Even for the same data, valid messages cannot be recorded and replayed, since the freshness value in the receiver will be different
- For regular CAN, only 27 bits used for MAC
  - May be brute forceable

Analysis of SecOC

- What happens if we have to replace a node, or a node misses a message?
  - Freshness values can become “de-synced”
- Need a way to synchronize freshness values
  - This can be OEM/implementation dependent
- Synchronization of freshness values can introduce stateful data that has to be maintained between nodes (across power cycles)

Synchronization

- Dealing with de-sync is left up to each OEM
  - This “secret sauce” is probably the first area to look at when pentesting
- De-sync can happen even in non-adversarial conditions
  - CAN errors can happen due to environmental conditions, buggy ECU startup code, etc.
  - Not all ECUs may take the same time to boot up, or may only turn on conditionally
- Most basic sync strategy (suggested by the spec) is to periodically transmit the full freshness value

Unforeseen Consequences

- If ECUs are de-synced there will be a soft fail
  - Normal CAN has deterministic failure
  - Receiving node may be de-synced and the transmitter would not know this
  - Can't do one-off messages without some kind of acknowledgment
- Simplest sync strategy (periodically transmit full freshness) is vulnerable to brute force
  - Solution is to require multiple correct MACs before re-syncing
    - — Exacerbates the "soft fail" scenarios

Key Management

- Achilles heel of a cryptosystem: key management. How are keys generated and stored?
- Potential solution: One global key
  - Simplest, can replace nodes with no configuration Key leaks once the whole system is caput
- Potential solution: One key per vehicle
  - Have to preload key to replace a module
- If an attacker gains code execution on one module, they can send any message
  - Only protects against unauthorized transmission from a rogue OBD/hardwired device (e.g. the Progressive hack)

Key Management

- Potential solution: one key per message
  - Ideal in asymmetric mode, but this is too slow to be practical
  - In symmetric mode all receivers of a message could become rogue transmitters
  - Requires lots of keys
- Possible compromise: “partial networking”
  - One key per logical “group” of messages

Apply

- Even with keys for each message, would SecOC protect against the Uconnect hack?
  - Attacker would still be able to send those messages the head unit could send and receive (i.e. those it needed the keys for)
- Take away: RCE (remote code execution) will almost always be a full compromise
- Modules should adhere to separation of concerns/least privilege principles
  - Code for playing MP3s should not be able to send CAN frames
  - If using a multitasking OS like Linux, isolate processes
  - Use a secondary processor for sending messages with a defined interface
    - Uconnect did this, but unauthenticated firmware updates rendered it useless

The End!

Contacting me

Email: jeffq@intrepidcs.com

Twitter: @realjeffq

Website: http://jeffq.com

http://jeffq.com/rsa_secoc.pptx