LEZIONE 18 09 2019 (slide 02)

How to secure something? It is impossibile to make a system impossible to attack, the only thing we can do is make the system very hard to attack. (if i don’t lock my bike, thief steal my bike). They just try standard attacks, if i am vulnerable of standard attack, i’m in big trouble. There are specialized attacks too, they spend much time and money to get in other’s systems.

Software is vulnerable, **vulnearbility** is a special kind of bug, it is not just annoyising, but it is used by attackers. Es: when i get my brench ( più if ) i implement software and i have lines of code and it is executed in the cpu. When i have a bug the program crashes or goes to a complete different state: software starts to execute something it was not programmed for, the attacker can decide what input to give to my software and he can deviate the regular flow of the software. They can execute whatever they want in system: **arbitrary code execution.**

Esempio:

se ho un vettore di 500 e voglio scriverci più di 500 dati non ho nessuno controllo -> overflow. Viene copiato nella memoria vicino al mio vettore, quindi se ho informazioni sensibili vicino al vettore, vengono sovrascritte. There is a particular velue called istruction point that is a pointer to the next istruction, by overwriting the buffer i overwrite the istruction point, in pratica se scrivo il carattere A che è 41 in ASCII, allora il mio istruction point leggerà 41 e andrà ad eseguire l’istruzione all’indirizzo 41.

L’esempio serve per dire che anche il software più semplice ha delle vulnerabilità che possono essere utilizzate dagli attackers.

Fuzzing is a way to test a software with a lot of inputs, oftent is used to make reverse engeneering, to generate the code and find vulnerabilities.

Defending the perimeter of a network (Candy Coach Security) is necessary but not sufficient, **firewall** is something to limit the traffic from inside to outside and viceversa, usually is put in the perimeter: it is a **perimeter defense.** As soon as an attacker gets control of one of the devices in the network, they can reach any device and execute what they want. Each pc needs its own firewall, that’s why every os is equipeed of a firewall. If something wrong is happening, it is necesary to detect to give awareness, there are also intrusion prevesion systems like antivirus.

The trend is moving to isolated networks in cars.

The scenario of a vehicle is very different by ICT world, there are much more costraints and it is a hard real time environment, the type of crypttography is different.

COMMENTS ON SLIDES

**OVERALL VIEW ON SECURITY**

It is a lifecycle: Cyber Defense Cycle: Prevention->monitoring-> detection -> analysis-> reaction -> response -> lesson learned -> prevention

I can monitor system online and offline. Off line analysis is better because i move datas out of the car and i can make the data be analized by forensics and experts.

* MONITORING: NIDS module can be integrated into a central ECU or multiple IDS modules can be integrated into multiple ECU, the HIDS moudule is implemented on each ECU, but in reality it is done onlu in the safety relevant ones.
* DETECTION: not so easy, person must have a knoledge base on cyber security, automotiv industry has not a precision definition of a virus in automotive domain, ECU behaviour is well documented, based on them a set of detection rules can be created. Another solution is if there is someone who knows VERY WELL how a car works, if he see something strange: anomaly detection can produce a lor of false positives.
* ANALYSIS: once anomaly is detected, there is a typical triage phase, type of anomaly and severity of it must be recognised. ( the attack is stealing mp3 files or preventing me to brake? ).
* IMMEDIATE REACTION: turn car off, depends on attack.
* OFFLINE ANALYSIS: What it must be to be done is off line analysis: store the anomaly and record, send them to experts (**PSIRT**). Forensics are mandatory in case of accidents. Black boxes are not yet equipped with component suitable to Court Criminale Procedure (they don’t have enough protection of their data). PSIRT ON SLIDES.

The attack can be passive or active (slide)

**ATTACK SURFACE ON MODERN VEHICLE**

**Attack surface** is sum of all **attack vectors**, an attack vector is any entry point (communication channel that an attacker use to interact with the syestem). Doesn’t mean that is easy to enter or that a vulnearability exisists.

**Trust boundary**: the boundary of a system, it includes subsytems that are considered to be trusted and that we want to defend from external attack. I am worried about any communication that crosses boundaries. Whenever an interaction crosses a trust boundary we have an tacck vector. An explicit enumaration of data flow can help. We can apply these concepts to automotive domain.

Internal: can bus, usb, infotaiment, obd connector <- | VEHICLE | -> external: cellular, wifi, bluetooth

(can bus is accesible only phisically)

**High-level threats**: remote control, steal vehicle, unlock, spy vehicle occupants, track a vehicle, install malware on the vehicle, …

**Drill down** by splitting the complex system into multiple components that are sill complex systems, attack vectors that crosse more boundaries may lead to more severe threats. TPMS (bluetooth) attacker just need to near the vehicle. Threats are classified based on the attack vector: Cellular, wi.fi, … , CAN

**CELLULAR**

: access the internal vehicle network from anywhere, expolit app in the infotaiment system and handles incoming calls; access SIM through infotaiment system, jam distress calls, track vehicles movements, OnStar. An attack based

ES: **OnStar system:** if you have an accidenti t sends data trough gps to a company, there is an application which let us do many interesting stuffs. What the application do is reaching General Motor servers and GM server is authorized through the internt to handle your vehicle. A 29 years old hacker took control over GM cars and was able to unlock any car which had OnStar system. http**s** sistes have digital certificates so we are sure that are trusted sites which we send http requests (encrypted communications). How to exploit it?

Step 1) place a device nearby the car that spoofs a SSID known to the mobile phone of your victim (like UNIMORE), can be generated on-demand by sniffing at wifi probes.

Step 2) …

**ES: 2014 Jeep Cherokee**

Vehicle is connected to the sprint which is connected to the sprint -> internet -> web server. Basic idea is “is it possibile to reach ip 62.xxx from the outside? If the attacker is inside the sprint nework he can directly connect to the vehicle. Sprint had firewall but it was allowed to connect vehicles. The sprint is 21.x.x.x and 25.x.x.x (class A networks), if i am in the sprint network i can get any other ip and check if there are connection listening on the port 6667.

**WiFi**

What happens if my car sets up a wi fi connection which i can use. In most of cases infotaiment system creates wi fi network. It is possible install malicious code on the infotaiment system, track vehicle (through MAC address), …

Wi fi attacks are not car specific (jamming, death, cracking). SSH server with weak/know credentials.

Citroen DS5 connects the devices on port 23 WITHOUT ANY AUTHENTICATION.

**KES (Keyless entry system)**

Key fob must be near enough, what can be done? Drain power from the key fobe, immobilezer to drain the car battery, lock out a key, capture crypto info leaked during nadshake (insicure protocols) due to very cheap electronic components, clone key fob, jam (interference) the key fob signal. Attacks to kes: atracks enables by **weak crpyto ->** OEM must recall all vehicles and change ecus. With the hardest crypto the system is 100% secure: wrong assumption. I guarantee that to use KES is secure if i use LOW POWER TRASMISSION: car opens only if i am very near => there are vehicles that can be attacked by RELAY: make the car think that the key fobe is near (no password,…).

**TPMS: tire pressure monitoring system**

Tpms are thought to last for all the llife of the car, the don’t use any cryptography.

Send impossibile condition to ecu casuing a fault, trick the ecu to overcorrect, track a vehicle based on the tpms unique id, and other stuffs, …

**Infotaiment**

Put console into debug mode, alter diagnostic settings, **use a malicious application to access the internal can bus** or to spoof data displayed to the user, install malware to the console. Access to micrphone, speakers, navigation system and other stuffs…

Esempio: audi uses Renesas chip to connect the can bus

**USB**

Usb is used to listen music generally. The dangereous thing is that i can an install modified update software on the vehicle. Attach a malicous usb with specially crafted files can break infotaiment system. Mazda is vulnerable.

**Bluetooth**

**CAN**

…

In the next lectures we will see how attack can bus.

**IN CAR VEHICLE NETWORK**

**Network**: ensamble of edges and nodes in which groups of nodes can exchanges information via a transportation medium. In motor vehicles we have:

* ECM (engine control module), ESP (electronic stability program)
* Simple control units: sensors, actuators
* Transport medium: data bus

**TOPOLOGIES (structure of the network)**

* **BUS**

Single element to which all nodes are phisically connected, it is easy to add other nodes. Info trasmitted in the form of messages and distributed over the bus. Need for access control mechanism. If a node fails, other can continue to work; the network fails ⬄the main line is broken.

* **STAR**

Data is exchanged between the individual node and the main node, which processes and realys information. Passive star: main node marely connects the bus lines of the network subscribers together. If a network subscriber fails or a connecting line to main node is defective, the rest of the network continue to operate. If a main node node fails => the full network doesn’t work (very bad for safety in automotive domain: single point of failure).

* **RING**

Each node is connected to its two neighbours. Single ring: unidirectional data transfer. The data is checked when it is received. If the data is not for the station, it repeats the message to other nodes. If a data arrives to the point of origin, it is discarded.

Double ring:…

* **MESH**

Every node is connected is to one or more nodes, not used in automotive domain. The cost is high, cost of the wire (and of the interfaces) =>radio uses mesh types.

* **HYBRID**

Network combined using multiple basic topologies (star-bus or star-ring).

**ADDRESSING**

Node has to transmit a message, how i decide the receiver?

An approcch is the **Subscriber-oriented**: the message contains the data and the identifier of the destination node, all receivers campare their id to the destion address, only the intented receiver reads the message.

In the automotive domain we use the **MESSAGE-ORIENTED**: the message contains a message identifier that identifies the class or type of the message, not an unique id. The transmitter doesn’t know anything about the destination. Each node receives the message, each receiver decides if it has to process (read) the message or not (similar to publisher-subscriber approach (MQTT)).

**Trasmission oriented**: just send data without any id. the destination is determinated by some charateristic of the network (time window).

**BUS ACCESS**

**Predictable vs random** (main distinction). Predicatble: bus access is determinated by network charateristics, only one node can trasmit at time -> collision free. Bus access determinated before vus access. Random: any node can try to transmit data if the bus appears to be free. Risk of collission -> retrasmission.

* **TDMA (time division)**: determinstic access method. Each node is assigned a time window in which it is allowed to transmit (fixed time schedule), no need for a controller but requires for syncronization and very precise timers. If i have a time window but nothing to transmit => time window is wasted
* **FDMA (frequency division)**: deterministic access method, each node is assigned a frequency in which it is alloed to transmit -> fixed time schedule, no need for a controller but requiers the ability to listen many frequencies.
* **MASTER-SLAVE**: master nodes the communication frequence, a node speakes only if it is spoken by the master. If the master fails, the whole network breakdown. Some master-slave protocols allow a slave to poll a master in order to transmit a message.
* **MULTIMASTER**: several nodes can access the tranport medium independetly. Bus access is uncontrolled -> collisions are possibile => collision detection and handling must be in place (no case of CAN bus): an example is the priority (car makers decides the priorities of the messages). If a node is broken the network continues to work and there is no need of precise timers and of the all bandwidht available.

**MESSAGE TIMING**

When a ECU wants to talk?

* **Event based messaging:** something that happens in engine or when someone press a button, another example is “the sensor releaves a condition”. If a node wants to talk but the bus is occupied => the trasmissioni s delayed (if the priority is low the message can be delayed and delayed..), these delayes become problematici f the bus becomes overloaded by a large number of network subscirbers which want to transmit. Pro:flexibility, good response to asyncronus events, only event that occured wants to transmit. Cons: non deterministic => can’t proof that a message was transmitted at the right time.
* **X-by-wire:** is the trend of the moment, mechanical and hydraulic systems are being replaced with electronic systems. It needs high requirements for reliability, safety and failure tolerance: latency for ciritical messages must be the smallest possible, the failure of a node must affect the network the less possible. Systems architectures for real time applications meet these requirements (predictable and verifable beahviour). Applications: steer by wire, brake by wire, throttle by wire, shift by wire.
* **Time based messaging:** (NOT TO USE): sequentially trasmission according to the schedule. Once all the nodes have spoken, the cycle restarts. Missing messages are immediatly detected. Pro: deterministic system, timely data transmission, detection and isolation of defective nodes. Cons: requires precise timers.

In a modern vehicle most of these tecnologies are toghther, need to be able to integrate subsytems.

**AUTOMOTIVE NETWORKING**

Safety (fault tolerance, ecc.. ), cost (choose cheapest solutions which satisfies the safety requirements) and security matters in automotive domain.

**Advantages of the bus systems:**

* Reduced costs with less weight
* Better reliability due to few plug in connections
* Simplification of vehicle assembling during production
* Multiple use of sensor signals
* Simple connection of a component to a bus

Requirements are:

- data transfer rate

- real time capability: guarantee that results are calculted within time costraints. The time costraints depends on the application.

- number of nodes

- interference immunity: to electromagnetic effects, depends on the safety relecance of electronics systems, possible solutions are checksum, other mechanism for error detection, or optical buses.

- Nothing about security

**Classification of bus systems:** proposed by SAE (society of automotive engineers), five classes, from A (slower and cheaper) to D (faster and more expansive). Slide 44. Class C is high-speed can bus and applications are: ABS, transmission control, ..

Diffrent application requires differents and incompatible bus systems => need for gateways (centralized or distributed (one gateway for each domain). Slide 50.).

**Automotive signal types:**

* Engine operating conditions
* Physical mesaurements recorded by sensors
* Control signals for activating servomotors
* Control element switch positions

An example is driving speed, engine speed, windshield wiper.

**CAN BUS – PART 1**

Important dates on CAN BUS history:

1991 : it is possible to use 11 or 29 bits as identifiers

1992: association to industrial automation and the first car was equipped with it

1994: pubblication of SAE J1939

2000: developement of TTCAN protocol

2004: ISO 11898-4

2011: starting of developement of CAN FD

2015: publication of

There are no open circuits, terminates with a 120 omega resistance (in parallel), so 60 omega between CAN H and CAN L.

There are many application of can bus beside automtive, like trucks, tractors, trains, off roads, maritime vehicle(standard NMEA 2000), industrial automation, home automation and also medical devices.

We have two types of CAN:

**High speed CAN:**

**How does it works?** Recessive -> deltaV = 0V beteeen canh and can low (both at 2.5V), dominant -> deltaV = 2V can h = 3.5V and can low = 1.5V.

**Low speed CAN**

Here we have completely different way to work, recessive deltaV = 5V ( 5 – 0) and dominanti s 2.2V (3.6 – 1.4).

The CAN node is composed of:

* Microcontroller, which interfaces with sensors and actuators. <- level 3..7 of ISO/OSI stack
* Can controller <- level 2 of ISO/OSI stack
* Can transreceiver <- level 1 of ISO/OSI stack

**Main characteristic:**

* Multimaster: all ECUs are equal: no need for a supernode
* Content-based addressing: messages include a label that identify the contents, ECUs are not given a specific address, they need to share the same set of labels
* Bus access based on arbitration: try to avoid collisions, in case of collision message with the higher priority (lowest ID) goes in the bus. Priority based on the labels.

**Arbitration**

1 -> recessive

0 -> dominant

When the bus is silent, it transmits a recessive level. All CAN frames start with a dominant level (by listening the bus all ECUs know that someone started to transmit), if 2 ECUs trnsmit the same level it is ok. If one ECU transmit a 0 and the other send 1 => the dominant level wins and the one which is transmitting recessive stops immediately. For all the other ECU, there is no collision, just the dominant level.

Result arbitration depends on the first bits of the CAN message, in call CAN frames at the beginning of the message there is message label, usually referred as ID => if you want win the arbitration just use only dominant bits.

**CAN Frames (how many kinds):**

* Data frame: most common
* Remote frame
* Errore frame: never see
* Overload frame: never see

**CAN message (slide)**

**Data vs remote**

A data frame trasmits actual frames, a remote frame asks someone to transmit data, in this case only the ECUU responsible of for emitting data frames with the same ID of the remote frame should answer. RTR flag is a part of arbitration part.

Remote frame are included but THEY SHOULD NEVER BE USED (might have collisions).

**Data frame field by filed on slide.**

MSb is most significant bit

**CAN 2.0B**

11 bits ID are enough for most automotive and industrial applications, but not for trucks, off-roads => solution is extend ID to 29 bit (18 bit more). This new type of CAN is becoming to be used in the automotive domain.

**Bit stuffing** (important)

There is a problem: all ECUs need to be syncornized to understand when a bit ends and then next bit begins, the issue: sender creates a message with 30 consecutive bits at the same level, so the receiver might be confused on how many they really are. Bad syncronization due to cheap clocks made of bad quartz.

Solution is to chenge the level every few consecutive bits, indipendently of message content ( 5 is the number in the CAN bus). A bit added for stuffing may cause the need for an additional stuffing in the next block of contiguous bits

**ERROR FRAME**

There are many methods to detect errors by ECU:

* Bit monitoring: listen while talking, there is errore if i listen something different
* Checksum
* Violation of bit stuffing
* Frame consistency check
* Acknowledgement

Usually i send errore frame as soon i detect an error, error flag is made of 6 dominant bits: 6 violates bit stuffing!. Error delimiter: Dominant bit will surely received by other ECU. 8 recessive bits.

**Overload frame**

Similar to the errore, but doesn’t notify Ecus, but it is used to delay the trasmission of the other frames on the bus. Also used to report a few corner case errors (reactive overload frame). Only transmitted betweenData frames or Remote frames.

**Fault confinement**

CAN bus is designed to be secure because it was made by mechanichals and electronics.

Problem: What happens if a faulty ECU continues to send error frames on the bus? It can be high-priority error messages, faults can be permanent or intermittent.

Solution: Ecu with permanent fault must be identified and after identify must be removed from the bus (logical remove, not physical).

All CAN transreceiver manage two counters for transmitting: TEC and REC.

**ECU states**

* Error active: REC <= 127 AND TEC <= 127 (good state)
* Error Passive ( REC > 127 OR TEC > 127) AND (TEC <= 255)
* Bus off (TEC > 255)

An attacker can increment counter and bring the ECU in bus off state.

Defenders can try do drive an attacking ECU into bus-off state possible reaction to attacks detected on the CAN bus. Attacker can try to drive a normal ECU into bus-off state allows attackers to shut down and impersonate any ECU.

The main idea is to cause many error by repeatdly by interfiering or overwriting the message of the victim.

There are two phases, depending on the error state:

…

**ATTACK OVER THE CAN BUS**

**Example of attacks that:**

**Do not** require knoledge about the target: bus off, fuzzing (used to analize how the system works), Denial of Service (DoS), injection of malformed frames, replay/spoofing … These types of attack are sufficient just to make a car not to work properly.

More sofisticated attacks -> **Require** knoledge about the target: attacks to safety relevants subsystems (park assists), attack ECUs by abusing diagnostic capabilities

**Denial of Service**

Attack consist of keeping can bus full by trasmitting high prioirity messages at highest data rate => lights do not turn off and engine doesn’t start: **attacked car is not utilizable**

**Fuzzing**

Inject lots of frames with random ID.. slide

**ECU spoofing**

Attack consist of pretending to be the ECM (engine control module -> rpm gauge to 500 ropm, general alarms, unusual vibrations; another attack: pretend to be the body control module -> doors unlocked, collision warning, activation of the electronic parking brake.

**Attack to the Adaptive Cruise Control (ACC)**

**Safety Relevant attack**: inject the same messages that are commonly used to increase the speed while ACC is active -> attackers can increase vehicle speed

**Advanced attacks: blocking legitimate attacks:** normally injected frames will be interleaved with normal data frames => it’s harder to perform many attacks. A possible solution is to disable (by driving it to bus off or by using diagnostics to start firmware update) the ECU we aim to impersonate before starting the attack.

Tutta la parte dei paper che spiega gli attacchi (da fare sulle slide)

..

Next step is attack from outside. Critical resources are data, not exposed, as the can bus in a vehicle -> so we need to find a way from another continent/network to get inside the network. When i am in -> i can move easy in the whole network. Lateral movement means that i start to compromise other systems inside the network. Once attacker gets data he can damage data, destruct them.

This also true in the automotive domain. In 09 attacks we started from internal recon (access directly to can) -> we already were in the network (that’s why oems didn’t care). Next step is to demostrate that in vehicle network are not isolated (easy); identify vulnerabilities in the interconnections, and much more…

Security pre-requisites:

* Buffer overflow
* Portscanning: related how system expose ports

**SECURING IN VEHICLE NETWORKS**

**PREVENTION**

Main principles: segmentation, segregation, defence in depth.

**Segmentation:** not having a flat network, but a hiercachical one-> no one node must be conncted to the others. Current solution: segmentation mostly driven by technical requirements: divide netowrks by application domain. (es:

ECUs that require fast networks are on high-speed CAN bus / Flexray

•ECUs that make do with slow networks are on cheaper low-speed CAN or LIN

•Everything related to multimedia is on MOST

Segmentation alone is not enough, attackers can still move laterally by crossing bridges among different segments. Segmentation allows to identify these bridges and put controls within them. There is limitate number of the segment we can put in a vehicle, we could Cluster together resources based on criticality and external connectivity and Assign each cluster to a different segment. Importanti is not to put critical systems on the same segment with external attack vectors.

To **segregate** = to enforce security policies and security assumptions.

E.g. we assume that no CAN dataframeshould be received from the OBD-II port (with the exclusion of UDS and other diagnostic protocols) there should be hardware and software in place to make sure that “unauthorized” traffic is blocked. Through segregation we define different “security domains”, each with its own security policies. Network-based segregation = limiting traffic. Need to have filterting rules to make only a set of communication to cross the checkpoint. Filtering rules should be defined based on security policies (again) and formal specifications. They should not impact legitimate functions. We must segregate at least at the perymeter of our system. Proper filtering rules prevent some system from working properly one or more ECUs are violating formal specifications.

Of course perimetral defense is not enough -> **Defense in depth :**

We need multiple layers of defense and they need to mitigate mistakes/holes in one layer, the effect of an inital compromise and the effects of the threats. Usually LANs use firewalls, in the automotive domain there is a recent concept of secure gateway. Simply put a secure gateway is just an automotive firewall. The firewall should be the only point of contact between two different security domains, it has to be properly configured through appropriate filtering rules based on well-thought security policies; it becomes a critical system -> it has to be secure (Are there any exposed configuration/update/command interfaces? Or Is it exploitable through overflows or other software vulnerabilities that affect the packet parsing and management logic?).

**Stateless vs stateful firewalls:** Stateless filters do not mantain an internal state, the final decision is based only on the input packet and the filtering rules. Stateful filters maintain an internal state: final decision affected by other packets previosly analized by the filter.

On what info i decide what packets can pass and what other must me blocked? The simplest possible choice is to open standard: Structure of the CAN dataframe: id, dlc, data as an opaque data structure, checksum of the frame (not within the payload), ..

DBC ( if i am a car maker)

•Know all (or some) proprietary specifications

•Signals (syntax, semantic, bounds), counters, checksum

•Proprietary diagnostic sessions, update procedures, …

Stateless and open standards: rejects unknown IDs, IDs that are known but that should be generated by a different segment (coarse grained), frames with incorrect (link-level) checksum (in can standard) or DLC.

* Stateful and open standard: can track the state of ISO-TP sessions…
* Stateless and DBC: Reject messages based on application level checksum and signal values, Reject IDs that are known but that should be generated by a different segment (fine-grained).
* Stateful and DBC: Reject messages based on application level checksum, signal values and counters, In principle: reject messages based on the vehicle state, Could do very complex things that require to compute the vehicle dynamics and the state of vehicle subsystems.

**MONITORING + DETECTION**

Intrusion: Any illicit activity aiming to degrade security performed by an attacker.

In automotive domain confidentiality is not that relevant: we don’t need to encrypt messages over the can bus. But integrity, availability, auth are important!

Intrusion detection system: software/hardware element that monitor activities to identify intrusions.

What an intrusion detection does: Generally speaking, the process of identifying intrusions by observing activities within a system.

Two main principles:

* Can observe system activities (not easy: can see only the nodes which are directly connected).
* Normal activities and intrusions are distinguishable (sometimes i can’t distinguish these attacks because the behaviour of the vehicle is normal).

**Intrusion Detection System** is a new concept for in-vehicle networks. It is legitimate to think to it as atypical IDS:

1.Gather CAN messages

2.Analyze messages looking for evidences of illicit activities

3.First goal: to log attacks

4.Second goal: to generate alerts (e.g., light or acoustic signal, haptic feedback)

5.Possible, third goal in IPS: to react (limit speed, bring to safe state)

by taking into account computational and cost constraints of ECUs, and no guarantee of OTA connections.

We have some issues with detection systems: Semantics of CAN messages is based on property information and are not standard, because it depends on car producer and vehicle model. There are many false positivies. Need for custom embedded devices to sniff and store CAN messages for analysis in a practical way.

We can calculate the precision and recall of detections in automotive domain.

Precision = tp / tp + fp

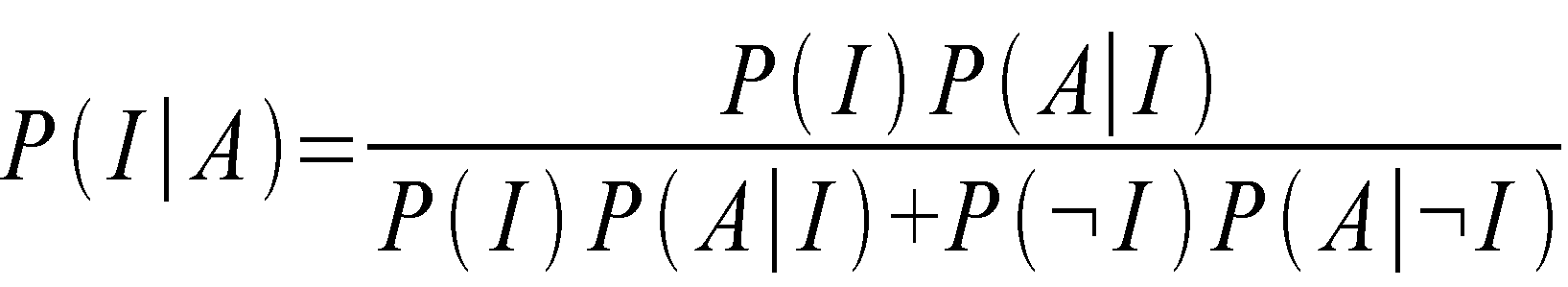
Recall = tp / tp + fn

In general the automotive domain is much more predictable than ICT domain -> it is easier to identify an attack beacuse the network is populated by ECUs. Hard thing is to get a good f1-score.

**Base rate fallacy:**

Problem is that the probably of that event is so low that i generate many false alarms.

How i compute P(Intrusion|Alert)? Through the Bayes rule



Need to reduce tha false positive rates. Try to “increase” P(I) by filtering out irrelevant events or focusing on low-speed channels.

**2 classifications for intrusion detection system: based on signatures and based on anomaly detection**

**IDS based on rules(signatures)**

A configuration is created and detection is based on static rules. Adaptation through new configuration and is applicable to known attacks, not applicable to zero days.

**Data-driven IDS(anomaly detection)**

Attacks not interest me. A model is created and the detection is based on statistics and machine learning (anomaly detection). Applicable to known and unknown attacks

**Vulnerabilty lifecycle**: vuln is within the software, there is time between the introduction of the vulnerability and until it is esploited to an attack. When security researchers find the vulnerability to the developers and fix it, but if the bad guys find it before reserches, they can perform attacks. After the vuln was disclosed, signatures of antivirus are updated, patch for the software is released and new exploit. In the time window “zero day attack” there is the first bad guy who finds the vulmnearbility, in that days there could be performed unknown attacks (no chances for signatures based ids to detected that attack).

**Anomaly Based IDS**

Gather data related to normal behavior. Create model of normal behavior. Detection mostly based on applications of statistics and machine learning. This kind of system know only what is normal: all the attacks are unknown to him. The IDS compares all events with the normal model:

Match ->everything looks fine here… no alert

No match -> anomaly found! Create alert (alert after an anomaly, not necessarily an attack)

If an attack cause no anomaly => the attack will not be detected. It is difficult to understand what attack cause an anomaly and if an anomaly is caused by an attack.

**Signatures vs anomaly**

* Signature-based IDS have to be constantly updated with new attack signatures, anomaly-based IDS have to be updated if the normal behavior changes
* Alerts generated by signature-based IDS are more useful for attack detection, alerts generated by anomaly-based IDS do not necessarily imply an attack, but allow to identify other anomalies that might be interesting
* If signatures are well written, they generate almost no false positive, almost
* If the system is verystable, also anomaly-based IDS generate few false positive. In general, anomaly-based IDS tend to generate more false positives than signature-based
* Signature-based approach cannot (by definition) identify zero-day attacks, they need a database of know attacks. Anomaly based IDS can identify a zero day, provided it causes a big-enough deviation from the normal model

In IT domain IDS are very common, there are no pure anomaly-based ids, but most of them are based on signatures -> it is a good idea to fuse both of them to have less uknown. (ML techniques use to associate a label to each packet).

How to evade an IDS? (make a false negative):

* Signature: find a zero day(which is hard) or start from a known attack, tweak it to make it just a little bit different from its signature (might be easy, depending on the nature of the attack and on how well written is its signature)
* Anomaly: Come out with an attack that does not cause large deviations from the “normal” model.
* Both: Hide real attack under millions of false alarms (very easy to do. Not exactly evasion, yet effective…) or attack the IDS.

In **automotive domain** IDS are resources costrained:

* They have to run on an ECU (possibly only using a portion of its computational
* Limitations on the implementable algorithms (sometimes only integer arithmetic, no “fancy” stuff such as logarithms and trigonometry)
* Much smaller throughput

There are few known attacks -> signature based IDS are useless!

Lack of public specifications (DBS files) for the automotive domain -> many approaches only applicable to features related to open standards

**Entropy based algorithm**

Assumption: Attacks are executed by injecting forged CAN messages in the CAN bus

Goal: to verify applicability and effectiveness of anomaly detection algorithms for intrusion detection

Tested anomaly detectionbased on **Message entropy**

Entropy (H) measures the amount of actual **information in** a dataset:

**Assumption: “Entropy of the CAN bus in normal conditions (with no attack) is stable”**

Intuitions:

* If the attacker replays an existing packet, then entropy should decrease
* If the attacker injects packets with random content (fuzzing), then entropy should increase
* **If entropy changes significantly, then entropy-based anomaly detection is effective**

Analisi ed esempi sulle slide

**Multi-tiered IDS approaches**

IDS can trigger an anomaly:

* in case of cyclic traffic, any extra traffic will change the frequency
* if message counters do not evolve in the expected way
* If the payload makes no sense (for example, does rpm toggle between 0 to 3000 every few seconds?)
* data is malformed (errors in the CRC field within the payload)
* …

Only high-rate attacks cause a significantly different bus utilization (250 attack packets per second)

**COURSE ON MODERN CRYPTOGRAPHY**

**Cryptography → how to transform messages to protect information**

The adversary has access to the secret information, but the text is transformed in such a way that the adversary cannot get the secret information (ne needs some information to partecipat the communication).

Cryptographic schemes can protect data on unprotected channels. Cryptographic schemes can provide additional protection on data maintained in protected channels.Crypt was born for data in motion.

Cryptographic protocols allow to protect data as a secure enclosure, the channel is insicure and it will stay ensecure.

In **symmetric setting** encryption key = decryption key,

in **asymettric setting** they are not the same, the encryption key is public, decrypt is secret.

Cryptographic primitives are mathematical tools, Cryptographic protocols are security-oriented communication protocols that leverage cryptograhpic primitives, which they include details at application level and details about key management.

**Symmetric encryption**

There are techniques based on anagrams -> ancient Greek’s scytale: easy to break.

The historical Caeserencryption system is based on a shift cipher

Sostituire le lettere con una permutazione casuale non va bene per la lingua italiana perché nel nostro linguaggio ci sono lettere che compaiono di più di altre (a,e,..), quindi sarebbe facile capire le corrispondenze.

The Vigenère cipher is a polyalphabetic cipher that can be seen as an improvement of Caesar cipher -> improve resistance against frequency analyses. The key is some word that is repeated to match the text that must be encrypted.

Kerckhoff sprinciple:

* Algorithm to be public
* Security relies on secretness of the key
* “A cipher must be practically, if not mathematically, indecipherable.”

Modern encryption scheme:

* the key space is large enough to prevent brute force search
* the scheme is designed to prevent cryptanalysis of the ciphertext (the scheme is designed to prevent cryptanalysis of the ciphertext)

Modern symmetric ciphers are designed for binary data, The base operation for symmetric crypto is the XOR -> it is impossible to infer information about the plaintext or the key from the output.

**One-time-pad (OTP)**

* XOR operation between the plaintext and the key
* size of the key = size of the plaintext
* the whole key is random

Perfect secrecy: no algorithms can ever break the encryption scheme, The size of the key must equal or greater than that of the message.

**Stream ciphers**

Encryption with stream ciphers is very similar to OTP, a bit-wise XOR operation computed between the plaintext and an intermediate key (keystream). The encryption routine takes a short random key as input. the PRG is a deterministic function that takes a small seed as input and that outputs pseudo-random data that cannot be distinguished from random. The encryption key and the nonce are used to generate the PRG seed. **encrypt(key, nonce, m): m ⴲPRG(key, nonce).**

**The DH key exchange protocol does not authenticate parties if a man in the middle attack oocurs. Alice** could sign her msseage with sign(g^a).

19 11 2019

**PUBLIC KEY INFRASTRUCTURE**

The PKI define how to securely distribute the public keys by using intermediate trusted parties. Each partecipant uses a key pair:

* Encryption: secret key to -> to dercypt, Public Key → to allow others to encrypt for you
* Signature: Secret Key → to sign, Public Key → to allow others to verify your messages

PKI is based on CERTIFICATES: a digitally signed document that binds some information to cryptographic material (standard x509).

Decryption is easy only for the owner of the secret key

(Root) Certification authority: someone who sign the certificate

Self signed certificate

**CSR:** Certificate signing request

**OCSP:** Online certificate status protocol

**CRL:** certificate revocation list

The trusted third party in PKI is the Certification Authority (both alice and bob trust this third part). The Certification Authority (CA) releases certificates that bind the public key to an entity (person, device, …). The certificates include additional metadata and information.

Delegating certificates: Requiring a few authorities to sign all certificates is not scalable => A hierarchical approach is a viable trade-off: the root CA certificates a CA that certificates a CA that…. A few CAs PK included in the software allow to verify a huge amount of certificates. (**chain of certificates**): una specie di proprietà transitiva della fiducia.

**X509**: Standard for PKI architecture, identifies the entity by using the distinguished name (DN). The certificate binds the public key of the entity to the DN and it includes other mandatory information.

Validating the certificate requires to

* verify the signature of the issuer
* verify all metadata accordingly to the application
* entity names (is it really the certificates for google.com?)
* validity (when is the expiration date?)
* type (I’m contacting a server: is this a valid server certificate?

•Validating all certificates in the certificate chain

•Verifying that the certificate is not included in the CA certificate revocation list

If we want to configure a private server for internal usage, we could avoid them and use self-signed certificates

How to get a certificate:

* we must issue a Certificate Sign Request (CSR)
* Create a valid key pair ( we don’t want CA to access our information)
* CA wants some money from us
* Securely store the secret key on the Web server

Any certificate has an expiration date but What if we need to revoke the certificate before the expiration date?

•Certificate Revocation Lists (CRL)

* Each CA publishes a CRL that includes all revoked certificates
* A client should control such CRLs

•Online Certificate Status Protocol (OCSP)

* An API to access CRLs to reduce overheads

**SECURING IN VEHICLE NETWORK (PART 2)**

No current application of encryption (any form of encryption) to CAN messages (as well as other in-vehicle networks) exist in licensed vehicles (lots of discussion about that within OEMs). In both cases, lack of open, standard crypto algorithms and approaches violation of Kerchoffslaw, (in)security through obscurity.

Two approaches for securing messages oberinsecure channels:

* Put message in a “secure envelope” [encrypt the message, send encrypted message normally] (encrypted emails).
* Create a “secure circuit” [virtual secure channel over insecure network, send plaintext message] (VPNs, HTTPS, …)

Can do both through symmetric crypto (Asymmetric crypto required to avoid Pre-Shared Keys (PSK))

Secure circuits usually:

* One to one communication -> no good in automotive domain because the CAN bus has a broadcast communication
* Require identification of the “secure” endpoints -> probably not suited for in-vehicle networks

**Secure envelopes on the CAN bus**

Possible goals:

1.Confidentiality ->not really relevant

2.Integrity -> relevant

3.Source authentication -> definitely relevant! ( if a message is generated by the distance sensori s generated we want that it was reallt it to generate it and not a random attacker)

4.Non-repudiation -> possibly relevant (attack attribution)

Version 1:

all ECUs connected to the same CAN segment already know the same PSK: We can satisfy 2 and 3 using MAC, will never be able to satisfy 4.

How does it work: attach a MAC to each message, verify the MAC when receiving the message. The gateway “translates” the MAC between different segments => Guarantee: anyone that does not know the PSK cannot communicate over the CAN bus.

Possible problems:

* if an attacker impersonates ECU x he will be able to inject message.
* Alghoritms requires at leats 28 bytes.

Solution: use HMAC (slide).

Is the PSK stored in cheap flash memory? -> easy to steal

Version 2

* each ECU has a couple of keys (pk, sk)
* All ECUs trust OEM’s Certification Authority

2 problems can be solved by using **asymmetric encryption.**

But also a digital certificate takes space (for can bus is not ideal) -> always possible to truncate it

Where i store key? In an asymmetric scheme, if I am able to steal the couple of keys of a given ECU, then I can communicate with all the ECUs that trust the same CA => i **really need** to store key in a secure memory and they cost a lot of money.

What if a pk is lost or stolen? Need to revoke the certificate. At least periodic checks about valid certificate should be performed and this is not easy.

**Dalla slide 17 in poi non si studia**

**SECURING ECUs (PART 1)**

Not only network myst be secured, but also ECUs. Main soruces of insecuring are:

* Bad design, not realistic threat model
* Bad implementation: software that contains vulnerabilities that can be exploited
* Bad configuration: exposed services, no auth for sensitive interactions
* Update procedures
* Bad management of crypto material: secret keys being accesible through diagnostic

**IMPROVING ECUs CYBERSECURITY:**

**HARDENING**

The process of making a system less vulnerable to attacks. Main steps:

* Minimize vulnerability surface, how to do? Close all exposed services and interfaces that are not necessary, Protect restricted interfaces through authentication and authorization procedures, Remove all unnecessary software
* Principle of least privilege: Identify user requirements, Limit user activities to the barely necessary, a normal user shouldn’t be the admin - > have all the privileges.

Same principles of the IT domain could be applied in automotive domain althought we don’t have a public checklist for embedded systems. (Checklist: change password every 60 days, …)

**ISOLATION**

Isolate domains, users, logical and physical resources.

Goal: mitigate attacks to a specific application/ software component. Might be achieved by hardware or software(virtualization).

* toward the consolidation of many functions in a lower number of more powerful (and hopefully more secure) ECUs

Hypervisor: it is able to run different operating systems -> it handles resources (VirtIO router)

Main issues:

* functional safety, need to demostrate that the component is safe (its behaviour)
* meeting the requirements of standards such as ISO 26262

Applications hosted on separate ECUs straightforward to test that ECU and its functions in isolation

**SECURE UPDATE**

Not a huge issue in the ICT domain: E.g., all windows updates are “secure updates”, meaning that the source is authenticated and update integrity and authenticity is checked before install. It’s a problem in the embedded systems domain: Resource constrained hardware, low memory, may or may not be directly or indirectly connected to the source of updates, Possible to execute crypto stuff? -> (low computional power -> hard to check integrity and auth).

**Firmware update**

Consumers rarely update their devices and Businesses don’t want the overhead of updating their devices.

Maintaining the software costs a lot to the businesses

This presents 5 types of problem:

1. Device resilience to power failure, network loss, etc.
2. Management of the authority to update devices
3. Privacy of the updates
4. Status monitoring of devices targeted by an update
5. Selection of which devices to update

Ideal: Receive a new firmware image from a trusted source and install it, boot into the new image and all works completely.

In real life: lose network connection, lose power, receive firmware that not work or corrupted (from untrusted source) or for a wrong device. Fail to boot new system, fail while controlling equipment.

**Resilience**

How we ensure that a device always works? there must be a piece of code that **cannot** be updated:

* **Bootloader**: it manages update, ensures that only a valid image is loaded. Device need to keep at least two bootable images so that one always works Bootloaders can be networked or non-networked.

Networked bootloader contains a netowrk driver, a network stack and a full update client, the bootloader: connects to an update server and download the update, after that it auth the update, installi t and hands over to the new image. Networked Bootloader are very complex -> they could be full of bugs, bootloaderneeds to execute crypto primitives and protocols to authenticate the update/source.

Static bootloaders are very simple and cannot be updated. The update client:

* downloads a new application image
* validates the image
* reboots

•Application images are big

•Static bootloadersstill need drivers if off-chip firmware images are used

•Sometimes, updates fail. The bootloadermust know how to revert a failed update

•The bootloadercannot be updated. No data or format used by the bootloadercan be changed

**Static vs networked**

They both have pro and cons:

2-stage boot:

Stage-1 static bootloader

Stage-2 networked bootloaders

•Recovery image

•Static bootloaderselects the regular image, or a recovery

•image that contains only the update client

Where to store update? On chip vs off chip

* On chip: on the main flash memory of the embedded device. The safest place to store an application image is on-chip. Any code that can modify the image could equally modify the bootloaderand remove any authenticity checks. The bootloadercan (and should) be very simple. On-chip flash is vastly more expensive than off-chip flash, Images can be stored at more than one location, so one of these strategies is necessary:

•Copy the candidate image to the active image location

•Execute the candidate in-place

* Off chip: on an external storage device. Off-chip storage is inexpensive (but not free) and plentiful. It allows a device to store many images. Simple access. It reduces flash cycles on the internal flash. The security isn’t trivial. Large storage space has challenges. Images can be stored at more than one location, so one of these strategies is necessary:
* •Copy the candidate image to the active image location
* •Execute the candidate in-placebootloaderneeds drivers to access external memory

Who wins? It is taks dependent

**Authority**

Authority in firmware update answers several questions (who wrote firmware? …)

Update over TLS:

Developers authenticate with the TLS server to start firmware updates (device trust server completely). The developer logs in to the update server and uploads a firmware; The update server decides whether or not to send the update, based on the developer’s permissions;

Issues: A centralized trust system creates a centralized point of failure…

Update with code signing:

Devices verify the firmware, not the connection, they trust a certificate that identifies the firmware author). An author can sign the firmware image before it is distributed.

•The devices trust the developer directly.

•The device verifies the signature of the firmware image before installing it.

•The risks posed by a centralized system are reduced because the author is trusted directly

•The author can perform signing on a very secure machine, such as a Hardware Security Module, which further reduces risk

Devices are now responsible for access control and authors are now responsible for the security, Devices must perform public key operations for each update and they are exposed to increased risk from old firmware. Devices must download the whole image before they can check the signature.

Dalla slide 30 sulle slide.

**SECURE BOOT**

a way to ensure that only authorizedsystem code runs on a device.

System code: pretty much anything up to the operating system. Also includes kernel modules

If image is corrupted, or you try to install your own (unauthorized) system code, boot will fail (or the module will not load)

Application code is (normally) not covered by secure boot mechanisms

Embedded systems generally include:

* Nand/nor moemory to store data
* CPU: processor for OS/apps
* DRAM: random access memory
* Interfaces: (Ethernet, wifi, CAN, UART, JTAG, …)

At power on: Processor comes out of reset, Begins running code from ROM or flash and Boot Loader (BL) is first non-ROM firmware to run. ROM/BL initializes HW. BL copied (by ROM or self) into DRAM before continuing, it continues hardware initialization from DRAM and validates, loads, and jumps into OS kernel. OS finishes init, goes to runtime steady state.

IoTdevices often have multiple processors/cores, These cores run distinct instruction streams (software), Frequently, they are DMA masters (Direct memory access (to ram)).

**Simple way to secure boot**

* On reset, processor starts from ROM
* ROM code loads/verifies bootloader , if invalid => alt
* Bootloaderloads/verifies OS and r/o filesystem(s) , if invalid => alt

After the boot Secure environment established, Everything inside of boundary is in known state, Can “trust” this system, It will behave in a predicable way, as expected… besides bug and software vulnerabilities…

**Advantages of secure boot**

Attack replaces some part of early boot code and takes full control of the system early on

* Robust secure boot can prevent this.

Of course, application may exploit system bug

**Secure boot with ARM platforms**

**ARM ATBSA**

TBSA: Arm Trusted Base System Architecture: reference architecture for the design and implementation of secure devices. TBSA encapsulates many best practice security principles implemented by providing hardware support for: root of trust, …

ARM-based systems might include one or more of these HW-based features

**Trusted Boot Board + UEFI Secure Boot**

TWO DISTINCT MECHANISMS : different Key/Certificates & PKI

SAME GOAL : verifying the authenticity and integrity of a software/firmware image before allowing its runtime execution

DIFFERENT TARGET IMAGES

Combined together they enable a full Secure Boot establishing a complete Chain Of Trust (despite different PKI) from the very first firmware executed up to the OS

**ARM TBB: PKI details (TRUSTED BOOT BOARD)**

A reference example on how to build a Chain of Trust from the very first ROM firmware executed (BL1) up to the first normal world firmware (BL33):

2 implicitly trusted components (tamper proof):

1. Root Of Trust Public Key (ROTPK) with SHA-256 hash stored on trusted registers
2. Boot Loader Stage 1 (BL1) stored on trusted ROM

2 Certificates pairs for each BL3x image

1) Key Certificate: holds the BL3x pub key needed to validate the corresponding Content Certificate

2) Content Certificate: holds the BL3x image hash to be verified against the hash of the loaded image

2 Key pairs used to sign/validate Key Certificates:

1. Trusted World Key pair (TW pub / priv) used for BL31 & BL32 Key Certificates
2. Normal World Key pair (NW pub / priv) used for BL33 Key Certificate

Public Keys and hashes are included as extensions to X.509 certificates

Certificates are self-signed: no need for a valid CA: ARM does not use certification authority

(authetnication flow on slide)

**UEFI SECURE BOOT**

2 pairs of keys

SLIDE