Fault injection on automotive diagnostic protocols

Bypassing the security of protected UDS implementations

Ramiro Pareja
Riscure Security Lab
pareja@riscure.com

Santiago Cordoba
Riscure Security Lab
cordobapellicer@riscure.com

Abstract

From the beginning of the era of electronics in vehicles, car manufacturers have been trying to simplify how to troubleshoot problems in their vehicles. The automotive industry has been improving diagnostic interfaces from the early attempts in the late 70's - based on proprietary technology which was often not even compatible between different models from the same manufacturer - to the modern and standardized buses and protocols. Nowadays it is uncommon to find a car that does not implement the OBD2 (On-Board Diagnosis 2) and the UDS (Unified Diagnostic Services) standards for diagnosis of the vehicle and the individual Electronic Controller Units (ECUs) respectively.

Due to the amount of information available through them, these diagnosis interfaces have been targeted by hackers and hobbyists from the very beginning. Modern protocols like UDS require authentication in order to access to critical assets (e.g. firmware). For years, attackers exploited trivial vulnerabilities in these diagnosis protocols to bypass this authentication, but state-of-art implementations make it impossible to simply logically bypass the security.

Our work presents fault injection as a technique to bypass the security of diagnosis protocol implementations that do not contain any logical vulnerabilities and therefore, that are protected against traditional logical attacks. This paper also illustrates the risk of a implementing a vulnerable diagnosis protocol, since it could serve as entry point for a scalable attack, and proposes some recommendations to mitigate the risk. Although this work is focused on the UDS protocol, similar approach could be taken to bypass the security of other diagnosis protocols like KWP2000.

Two different ECUs, both from car models currently available for sale, were tested against fault injection attacks. Our tests proved that it is possible for an attacker to inject faults and bypass the UDS authentication, obtaining access to the internal Flash and SRAM memories of the targets. By analysing the dumped firmware, the keys and algorithm that protect the UDS have also been extracted, giving full access to the diagnosis services without requiring the use of fault injection techniques.

1 UDS and its security

UDS is a CAN-based protocol for vehicle diagnostics based on the older KWP2000 protocol. Since its standardization with the ISO 14229 [1] , it has been progressively adopted by car manufacturers and has become the most common ECU diagnosis protocol present in modern cars. Initially designed to read Diagnostic Trouble Codes (DTCs), the protocol was extended to support access to the internal memory, sensors and actuators; executing testing routines; and downloading or upgrading the ECU firmware. The possibility of accessing the internal memory of the ECU makes the UDS protocol the perfect entry vector for hackers and attackers. Dumping the firmware could reveal secrets or exploitable vulnerabilities that allows the adversary to escalate the attack, either by

making it available to more persons (e.g. sharing keys) or by attacking the system remotely (e.g. a remote vulnerability that does not require physical access).

If the UDS implementation complies with the ISO specifications, access to the memory and firmware of the ECU is not directly allowed. A client (the tester in ISO14229 terminology) trying to access to a server (the ECU according to ISO14229) must establish special sessions for using security relevant services. For example, firmware upgrade services are typically only available when establishing a programming session. The different diagnosis sessions are selected by the client sending a request to the DiagnosticSessionControl service, but the server enables the session only after the client is authenticated by the SecurityAccess service.

The SecurityAccess service implements a challenge-response authentication scheme where the client requests a random seed to the server. This seed is encrypted in both sides - the client and the server - to generate the session key. If the server receives the expected key from the client, access to the diagnosis session is allowed. Figure 1 illustrates the authentication process.

The ISO14229 details the authentication process and the messages exchanged by both parties but no recommendations about the seed generation or key calculation are included in the standard. In the past, this leaded to multiple vulnerable implementations where small key sizes or broken Random Number Generators (RNG) allowed the attackers to brute-force the keys. Current state-of-the-art implementations are virtually free of logical vulnerabilities that allow an unauthorized user to bypass the UDS security. Nonetheless, these implementations could be vulnerable to hardware attacks like Side Channel Attacks (SCA) or Fault Injection (FI).

2 Hardware attacks on UDS

A Side Channel Attack is any attack that gains information from the target by exploiting information leaked from an otherwise correct implementation of the system [3]. A very powerful side-channel attack is Power Analysis in any of its variants (i.e. Simple, Differential, Correlational); these attacks use the power consumption of the chip to guess what operations are being run at a given moment. In the authentication process described for the SecurityAccess service, an attacker could use these Power Analysis techniques to identify which algorithm is being used and extract the keys. These attacks, expensive and slow in the past, can be executed nowadays in few hours - days at most - and with a restricted budget of few hundreds of dollars.

Side Channel Attacks may be an interesting topic for future research, but we do not discuss them further in this work. Instead, we consider another form of hardware attack.

Fault Injection attacks induce fast variations in the operating conditions of the chip to cause unintended behaviours - the faults - which could compromise the system security [2]. The most common techniques for injecting these faults are voltage glitching - a fast variation of the power supply voltage - and EM glitching - a strong and localized EM pulse that generates power variations in the internals of the chip. When carefully tuned and precisely timed, glitches can alter the execution flow of the software and bypass security checks. For example, an attacker can use a Fault Injection attack to bypass the key verification step on the UDS authentication process detailed in Figure 1.

Wiersma and Pareja [4] proved the effectiveness of this technique to attack ASIL-D automotive microcontrollers in semi-controlled environments. The work presented in this paper proves that fault injection is not only useful in a semi-controlled environment but also in black-box evaluations of final automotive applications running on real targets.

3 Targets

Two instrument clusters were chosen as targets for testing their UDS implementations against fault injection attacks. Both targets were selected based on their market relevance and representativeness of typical hardware. These ECUs come from car models that are currently available for sale and are very common in Europe. Out of all the ECUs present in a

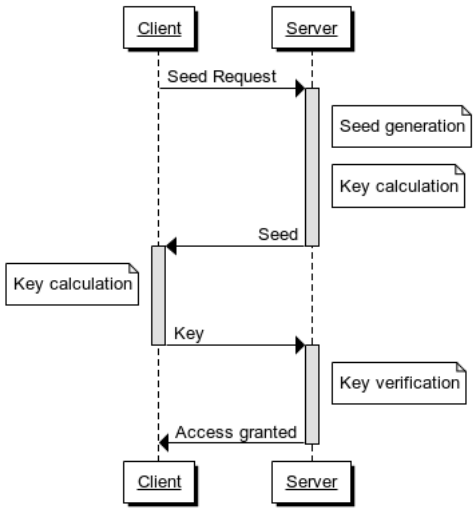

Figure 1: SecurityAccess flow

car, the instrument clusters were chosen for testing because their odometers are commonly targets of attacks. Changing the kilometre count on the instrument cluster could be the final goal of an adversary trying to gain access to the UDS services.

Target A is an instrument cluster from a B-segment car manufactured in 2010. The only information we found on the internet about this device was documentation of the pin-out of the ECU connector. By reversing the PCB, the main microcontroller was identified and its datasheet downloaded. Using this document, the power and reset pins of the microcontroller were identified and the PCB was modified to facilitate the injection of voltage glitches. This modification consists of isolating the chip from the PCB power plane to avoid the voltage glitches affecting other components. Additionally, the decoupling capacitors are removed, as they could flatten the voltage glitches.

Target B is an instrument cluster extracted from a C-segment car manufactured in 2015. No information is readily available on internet about this target or the chips identified in the PCB. Because the datasheet of the microcontroller is not available and the chip pin-out is unknown, EM fault injection is used instead of voltage glitching. The pin-out of the microcontroller could be identified by reversing the PCB; even so, EM glitching is used against this ECU in order to prove the convenience of this technique in situations where no information about the target is available.

Both targets have UDS implementations secured against logical attacks. The seed and the keys are 32-bits long and after 3 failed authentication attempts, a lock-down time of 10 minutes is imposed. The lock-down is persistent even if the power is removed or the chip is reset.

4 Bypassing the UDS authentication

4.1 Attacking the key verification

Target A was prepared for voltage fault injection as described in the previous section. The goal of the attack was to use fault injection to bypass the key verification in the SecurityAccess service. A custom device based on Arduino was programmed to act as the UDS client. This device is connected to the CAN bus of the Target A, and invokes the SecurityAccess

UDS service. When the client sends the key to the server for its verification, the client generates a signal that is used by the voltage glitcher device as a trigger to generate a voltage glitch.

The success of the glitch - in this case, the ability to bypass the key verification - depends on different factors like the voltage and duration of the glitch or the exact moment in time when it is injected. These factors are called glitch parameters and are normally found using a trial-and-error approach. A sequence of experiments - the fault injection campaigns - is run to try different sets of parameters. Progressive iterations of the experiments refine the parameters trying to optimize the success rate. This process is called characterization and normally requires thousands of glitching attempts. Ideally, the characterization is run on a target fully controlled by the attacker, where a special characterization software is executed. In situations where no custom software can be loaded in the target - typical in black-box scenarios - the characterization process and the optimal parameters search (parameter tuning) is done when running the final attack.

We run an initial fault injection campaign with Target A and a very broad set of parameters trying to bypass the UDS key verification process. We soon realized that due to the 10 minutes lock-down enforced after 3 authentication errors, it is not possible to run the fault injection campaign in a reasonable amount of time. A 10,000 glitching attempts campaign - the minimum to have any chance of succeeding - would take 23 days. We decide instead to look for alternative attacks on the UDS protocol.

5 Attacking the memory access services

Most of the modern UDS implementations enforce a lock-down time after failing the authentication process. They, however, do not enforce the lock-down if any other service is called without enough privileges. This opens the possibility of an attacker using fault injection to bypass the privileges check on other UDS services. The services ReadMemoryByAddress, WriteMemoryByAddress, RequestUpload, RequestDownload , TransferData and Routine control are potential targets for this attack. We focus only on the ReadMemoryByAddress, but similar approach could be used to attack the other services.

The service ReadMemoryByAddress allows the client to read the server memory. The address and number of bytes to read is passed as parameters. In the evaluated targets, the address parameter directly refers to the physical memory space of the microcontroller, but in other targets it might refers to a logical memory space instead. After invoking the service, the UDS server checks if the stablished session has enough privileges to access to the memory. If the client is authorized, the service returns the requested bytes to the client. Otherwise, an error code (NRC) is generated. Figure 2 shows a simplified diagram of the ReadMemoryByAddress invocation.

We prepared an attack on the Target A aiming to bypass the privilege level check with a voltage glitch injected when the ReadMemoryByAddress service is invoked. The exact moment when the privileges are check is unknown, but it is bounded to the period of time between the UDS petition sent by the client and the NRC response from the server. In the Target A, this window is 1700 microseconds approximately. We ran an initial fault injection campaign covering the entire window. After one day and 45,000 glitch attempts, we had some successful glitches indicating in which moment in time the privilege level is check. In order to maximize the success rate, we ran subsequent campaigns for characterizing the target and tuning the glitch parameters. The highest success rate obtained after the tuning process is approximately 3%.

The final attack was launched by glitching the ReadMemoryByAddress to read different memory addresses. After approximately 3 days and 300,000 glitch attempts, the full 512KB of the flash memory were successfully extracted from the microcontroller.

A similar attack was mounted on Target B, using EM glitches instead of voltage glitches. EM fault injection requires a longer characterization campaign because the surface of the microcontroller package needs to be scanned looking for the most sensitive spot for glitching. The highest success rate obtained

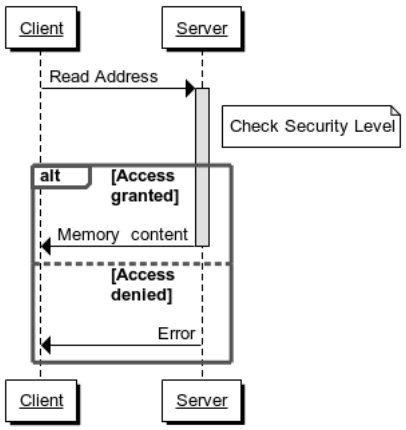

Figure 2: ReadMemoryByAddress flow

for the EM glitches after tuning was 1.7% and the full 3MB of flash memory was extracted after approximately 4 days and 500,000 glitch attempts.

6 Escalating the attack

6.1 Extracting the UDS keys

In security literature, an attack is said to be escalated when it is built up in terms of threated assets or reproducibility. The reproducibility of hardware attacks is generally lower than in software attacks as it requires the use of hardware tools and physical access to the target. In the specific case of the attack described in section 5, the reproducibility is low as it requires a complex glitching setup and taking the ECU apart from the car. Nonetheless, this attack can be escalated to obtain easier reproducibility by analyzing the extracted flash memory. This flash dump contains the firmware ran by the ECU, including the UDS services. By doing reverse engineering of the firmware, an attacker could identify the algorithm that generates the keys during the UDS authentication process and use it later to invoke the SecurityAccess service.

Due to the size of the extracted firmware, it is not easy to spot where the UDS routines are located. Reversing the entire firmware is a tedious and time-consuming task, so a smarter approach is needed. We use the NRC numbers to locate potential candidates for the UDS services. For example, the SecurityAccess service returns the NRC 0x35 if an invalid key is provided. By listing all the instructions where the number 0x35 is used as a literal and limiting the reverse engineering process to the code around those instructions, we dramatically reduce the number of functions that are potential candidates. For target A, only 160 candidates were found in the 512KB flash dump. Moreover, the candidate list can be reduced further if the assumption that other NRCs are located nearby in the code. In target A, the NRCs 0x22 (Conditions Not Correct), 0x12 (SubFunction Not Supported) and 0x24 (request Sequence Error) were found in the next 16 instructions after the NRC 0x35.

After identifying the SecurityAccess service in the binary, the authentication algorithm can be reversed and extracted. The algorithm is simple and uses a secret 16-bit number to generate the authentication keys. Both targets uses the same algorithm but they have a different secret number. It is unclear for us if these secret number are unique per ECU or per model. The algorithm was verified by using it after calling the SecureAccess service on the actual target

A, and was confirmed to generate the correct keys. Both targets use the same algorithm.

6.2 Gaining runtime control of the ECU

Having the key generation algorithm to authenticate the UDS client basically means to have full control of the ECU. Depending the implementation, reading or writing into the ECU memory can be more or less restricted. In all the evaluated targets it was possible to read and write the full memory space directly or indirectly. In one of the evaluated targets, uploading new firmware using the RequestUpload service requires encrypting the image with an AES key - which can be extracted from the flash dump - and signing it with a private RSA key - which cannot be obtained from the flash dump. The service WriteMemoryByAddress is however more flexible, as it can be used to write data directly into the entire memory space, including SRAM and peripheral registers. Flash memory is not directly writable, but it can be indirectly programmed by writing the Flash controller registers. In a different evaluated target, the WriteMemoryByAddress function was restricted to access a limited set of SRAM and register addresses used for debugging. These set of accessible registers includes the DMA controller registers, which indirectly allows an attacker to access to the entire memory space.

Gaining runtime control of the ECU is relatively simple if the attacker has write access to the SRAM memory. The most common way to obtain it is to fill an SRAM buffer with the payload to execute and overwrite the return address in the stack for forcing the CPU to jump into it. This approach was tested on target A. Other possible ways for gaining runtime control includes overwriting application code stored in the SRAM, manipulating the reset or interruption vectors, permanently writing the flash memory with the payload, etc.

6.3 Jumping to other ECUs

A full practical attack on a vehicle typically involves getting control of different ECUs. Depending what is the goal of the attacker, the threated assets might be in a different ECU than the one used as entry vector. For example, an attacker looking for crashing a car remotely would enter the car network through the infotainment system or the telematics unit and from there, he would attack the Electronic Break Control Module (EBCM). If both ECUs are connected to different CAN networks, the attacker might need also to attack the gateway in between. The attacker would need to find vulnerabilities for all the ECUs involved in the full attack. The same approach taken in section 5 and 6.1 can be used with all the ECUs involved, but to save time we tried a different approach.

In 6.1 we found that Target A and Target B uses the same algorithm to generate the UDS keys, but with a different secret 16-bit number. We assume that this algorithm is also used in all the other ECUs from the same manufacturer, but changing the 16-bit secret number. In order to confirm this hypothesis, we try to attack the Target C, a gateway from the same car model we extracted the Target B from.

We use brute-force to determine the secret 16-bit number. Due to the 10 minutes lock-down after 3 failed attempts, brute-forcing the 16-bit secret number would take 76 days on average. We identify a serial EEPROM in the gateway which is used, among others, to store the number of failed authentication attempts. In order to accelerate the brute force process, we externally manipulate the EEPROM signaling to prevent the MCU the store the failed attempts. This bypasses effectively the lock-down timer and the correct secret number was found in less than one day. After establishing a session with brute-forced secret number, calling any memory-related service returned an error. We did not research further, but it is possible that these services are not implemented or that they are only available using a different session number.

The information gained attacking the instrument cluster and the gateway could be enough to mount a full attack on the odometer and change the kilometres without having physical access to the ECUs. The attacker would have to first attack the gateway, which is directly connected to the ODB2 port under the driving wheel of the car. With runtime control of the gateway, the attacker could jump to the instrument cluster and modify the kilometres stored in the

memory. In some cars the kilometres are also backed up in a second place (e.g. engine ECU), so a third ECU should be attacked. We did not explore further this possible attack on the odometer.

7 Security consequences

The described attack affects the confidentiality of the firmware and keys present in ECUs. The most immediate consequence of it is the threat on the manufacturer's Intellectual Property. Developing modern car functionalities like ADAS (Advanced Driver Assistance Systems) or ADS (Autonomous Driving System) requires a lot of investment in money and time. Instead of developing its own solution, an unethical company could illicitly gain a competitive edge with a minimum investment by using fault injection to extract the firmware from a competitor's product and reverse engineering it. Malicious attackers could also benefit of reverse engineering the firmware to find secret keys (e.g. immobilizer or remote keys) or other vulnerabilities that could help to escalate the attack (e.g. remote vulnerabilities).

Bypassing the UDS authentication can also be used to affect he integrity of the firmware running in the ECU, either temporally - gaining runtime control - or permanently - reflashing the ECU. This has been extensively done for years by hobbyist and car enthusiasts that 'tune' the car seeking an increase of the performance or tampering the odometer. More sinister scenarios are also possible, like an attacker installing malware in the car to ask for a ransom or to cause a malfunction on the safety mechanism, causing injure or death to the passengers.

8 Conclusion

The experiment described in section 5 proves that it is possible to use fault injection to bypass the UDS security on an otherwise logically secure UDS implementation. Later, subsection 6.1 illustrated how a fault injection attack can be escalated by extracting the firmware and reversing the UDS authentication algorithm. After it, no more hardware attacks are needed to have full access to the UDS services, as the attacker can generate the correct authentication keys. Finally, subsection 6.2 showed how an attacker with writing access to the memory can gain runtime control of the ECU and subsection 6.3 showed how to escalate the attack to other ECUs. All these experiments demonstrate that fault injection attacks can be used to threat the ECU security and manufacturers should consider implementing countermeasures against them. These countermeasures have been extensively described in academic literature and proved their effectiveness in many products already deployed (e.g. smartcards, SoCs for the PayTV market, etc). Although many of these countermeasures can be unpractical for the automotive industry due to their added costs, others like software countermeasures can be easy and cheap to implement in the current ECUs [5].

9 Acknowledgments

The authors would like to thank our colleagues from Riscure for their support. In special, we would to thank Eloi Sanfelix, Alyssa Milburn, Niek Timmers, Prem Jagai, Justin Black, Ileana Buhan and Marc Witteman for their ideas, contributions and help that made this paper possible.

References

[1] ISO 14229, Road vehicles — Unified diagnostic services (UDS). `https://www.iso.org/standard/55283.html`.

[2] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. The Sorcerer's Apprentice Guide to Fault Attacks. *Proceedings of the IEEE 94*, 2 (Feb. 2006), 370–382.

[3] Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. 01 2007.

[4] Wiersma, N., and Pareja, R. Safety != security: On the resilience of ASIL-D certified microcontrollers against fault injection attacks. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017* (2017), IEEE Computer Society, pp. 9–16.

[5] Witteman, M. Secure application programming in the presence of side channel attacks. *https://www.riscure.com/publication/secure-application-programming-presence-side-channel-attacks/*.