Proc. of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)
16-18 November 2022, Maldives

On the Feasibility of Remotely Triggered Automotive Hardware Trojans

Athanasios Moschos
COEUS Center
Georgia Institute of Technology
Atlanta, GA, USA
amoschos@gatech.edu

Kevin Valakuzhy
COEUS Center
Georgia Institute of Technology
Atlanta, GA, USA
kvalakuzhy6@gatech.edu

Angelos D. Keromytis
COEUS Center
Georgia Institute of Technology
Atlanta, GA, USA
angelos@gatech.edu

***Abstract*—Modern vehicles are comprised of many separate computer systems running on Electronic Control Units, or ECUs. These ECUs allow for functionalities as diverse as advanced computer-aided safety features to browsing the web. However, the explosion of ECU-enabled capabilities has presented flaws that allow adversaries to literally stop vehicles in their tracks without physical access to said vehicles. Normally, causing of unintended behavior would require physical access to the ECU and/or exploitation of vulnerabilities present in the ECU's software.**

In this paper, we discuss how Hardware Trojans can act as the physical access intermediates to allow the remote triggering of malicious payloads embedded in ECUs, through seemingly benign wireless communication. We demonstrate a proof of concept ECU hardware trojan (HT) on a RISC-V based processor emulating an ECU. The HT takes advantage of benign radio functionality, emulated by TCP packet transmission, to provide a triggering pathway for disabling the ECU and thus, the host vehicle. This attack vector, enabled by deep and often opaque international supply chains common in the automotive industry, provides a stealthy way to conduct both targeted and fleet-wide remote attacks against vehicles.

***Index Terms*—Automotive, Electronic Control Units, Fabrication, RISC-V, Hardware Trojans**

I. Introduction

Electronic Control Units, or ECUs, constitute the cornerstones of modern vehicles, controlling a myriad of vital and leisure functionalities. Those functionalities range from the control of the brakes, the steering wheel or safety features, to the control of the vehicle's infotainment system and the various communication interfaces available (WiFi, 3G/4G cellular network, radio, Bluetooth). ECUs are usually interconnected through a CAN bus, providing a simple way of coordinating between different subsystems. Nevertheless, the level of this interconnection has also raised important security concerns. For example, it has been shown that an adversary with physical access to a vehicle's CAN bus can fake speedometer readings, unlock the doors, or even kill the engine [1]. However, the need for direct physical access cannot act as the saving grace for car manufacturers, as wireless communication channels have been also utilized to exploit software vulnerabilities on ECUs and induce unintended behavior [2], [3]. On top of that, no significant attention has been given to the consequences of potentially compromised hardware being included in modern automobiles, especially in the context of hardware trojans.

The introduction and increasing adoption of open Instruction Set Architectures (ISAs), like RISC-V, has lowered the entry barrier to chip manufacturing. This has lead to commercialization of automotive-grade RISC-V ECUs [4]–[6]. The open nature of the architecture these systems are based on enables a closer examination at the hardware level giving rise to the potential of HT attacks against them. HTs represent circuitry surreptitiously inserted into an ECU to implement functionality that the attacker wishes to introduce to the system, typically a means to remotely access or disable the target. The two key elements of an HT are triggering and payload. Triggering refers to the conditions and logic under which the HT will be activated, ideally only under the attacker's control; while payload refers to the intended effect that the attacker wishes to achieve (*e.g.,* disabling the ECU).

II. Our Contribution

In this paper the following contributions are made:

- A conceptual threat to the automobiles' industry is presented, which bridges the gap between the necessary physical and/or remote access needed by an attacker to tamper with an automobile's functionality. More specifically, the showcased scenario involves a HT that is inserted inside the Telematics Control Unit of an automobile. The HT can be controlled through wireless transmission of arbitrary TCP packets and can tamper with the automobile's functionality. The threat of hardware trojans has only been superficially explored so far in the context of ECUs [7]. Our trojan is implemented inside a RISC-V 64-bit CPU design that acts as the TCU.
- A HT triggering mechanism is demonstrated, that takes advantage of Vehicle-to-Vehicle (V2V) systems (*e.g.,* Dedicated Short-Range Communication (DSRC) radios,

This work was supported by the Office of Naval Research under Contract #N00014-19-1-2287.

Cellular Vehicle-to-Everything (C-V2X) solutions) expected to be present in most autonomous vehicles in the near future. In particular, our analysis focuses on the DSRC technology, due to its similarity with the WiFi technologies and for ease of experimentation. The QEMU emulator targeting RISC-V 64-bit CPU implementations is utilized, to unveil the instructions executed during the processing of an arbitrary TCP packet. Such a TCP packet may be received via a DSRC communication channel.

- The implemented HT monitors a specific General Purpose Register (GPR) and listens for special sequences of bytes processed by the RISC-V CPU. A special sequence of bytes introduced in a TCP packet and received over DSRC, enables the payload of the HT that asserts the global reset signal of the CPU. This in turn leads to the deactivation of the TCU.

III. Related Work

Existing works in the literature exploit both hardware and software to seize control of vehicles. Bozdal *et al.* use a HT implementation to perform a Denial of Service (DoS) attack on the CAN bus, a communication channel frequently used by critically important devices [7]. The authors rely on the CAN bus' inherent weakness of unauthenticated broadcasting, to impersonate different nodes on the bus and set them offline through erroneous packet transmission. To that extent, the authors hijack a node in a vehicle's CAN bus to implement their trojan. Their trojan masquerades as the node it wants to attack and initiates erroneous communication on this node's behalf. Once a certain threshold of erroneous packet transmission is reached, the node that the trojan is masquerading as, falls to a bus-off state and no longer serves requests from other nodes.

Nie *et al.* investigate wireless attacks on Tesla cars that lead to the compromise of in-vehicle systems [2]. Their work shows how a chain of vulnerabilities, including those found in the car's web browser and the Linux kernel running on the vehicle's Central Information Display (CID), ultimately allow an adversary to send arbitrary messages on the CAN bus. With this level of control, they are able to open the trunk or disable power steering and brakes, even while the car is moving.

Checkoway *et al.* contemplate the realism of threat models requiring physical access to perform attacks on automobiles [3]. In doing so, they investigate the susceptibility of automobiles to remote compromises, including through short-range and long-range wireless access. The authors discovered a range of threat vectors that permit malicious remote exploitation (*e.g.,* vehicle control) through ECUs possessing external interfaces (*e.g.,* Bluetooth, cellular data links). One of the ECUs possessing such interfaces is the TCU, or Telematic Control Unit. The TCU is one of the most crucial ECUs included in an automobile, as it is responsible for a plethora of safety and diagnostics features. Furthermore, telematics systems are often used to provide remote management services which require connection to control systems for the door locks and even the engine.

Koscher *et al.* use the telematics unit to bridge originally separated CAN bus networks, allowing low priority components (*e.g.,* the radio) to compromise high-importance systems controlling the engine or brakes [1]. With these details in mind, it should be of no surprise that the TCU is a prime target for an adversary.

IV. Threat Model

Our conceptual threat scenario pertains to the recent advancements in autonomous driving and the potential risks arising from it. Autonomous driving is becoming evident in self-driving trucks [8] and a disruption in their mobility can have damaging effects from local supply chains to country-level economies. Moreover, the immobilization of such an automobile fleet can prove damaging to the manufacturer's brand.

Nevertheless, to perform an attack that can immobilize a fleet at a large scale, both a physical and a remote access mechanism on the automobile itself are necessary. An HT inserted in an automobile's ECU at the foundry, can provide such a physical access mechanism at a large scale. In addition, an existing wireless communication channel can prove to be useful as the triggering channel for an extensive attack.

Our threat model assumes an attacker with the ability to insert a hardware trojan inside a TCU, during the TCU's fabrication. This special type of ECU possesses interfaces that handle the vehicle's wireless communications. Modern ECUs implement a variety of wireless communication protocols, as well as employ open-source software (*e.g.,* the Linux kernel) [3]. The attacker is assumed to have enough knowledge about the OS that will be running inside the TCU, as well as the software responsible for processing any wireless communications. An aspiring attacker can gain enough knowledge about them by reverse engineering the software/firmware used in previous generations of a specific ECU attack target [3].

In our scenario the attacker is exploiting the DSRC technology, that is used for V2V communication and is based on the 802.11p extension [9] of the 802.11 standard for wireless local area networks. Moreover, the data plane of the WAVE protocol stack includes support for the TCP [10], [11]. It is assumed that an attacker in the proximity of a target vehicle can wirelessly transmit a TCP packet that will be captured by the TCU of the target vehicle via the DSRC route. This TCP packet will include a malicious sequence of bytes, responsible for triggering the HT. The are no hard constraints in terms of what values this sequence of bytes can take, as even a TCP packet that will eventually be discarded will first be processed by the CPU.

While our threat scenario simulates an attack triggered via the WiFi-like DSRC interface of a TCU running the Linux kernel, neither the methodology nor the attack are protocol-specific. This type of attack can be conducted on any of the TCU's wireless network protocols, thus enabling the potential for attacks through longer-range communication channels, such as the cellular or satellite communication channels. As an example, attack coverage of a large geographical area can be

achieved through a satellite that will emit the triggering TCP packet(s) without a requirement for a bidirectional channel.

V. Attack Design Methodology

With the need for a wireless trigger in mind, locations in the CPU (*e.g.,* GPRs) that are influenced by the data bytes of TCP packets are identified. Next, a HT circuit is created that is activated by the reception of a TCP packet containing a specially crafted data byte sequence. Finally, the implemented HT is evaluated to have negligible chances of being triggered by random appearances of the chosen triggering data sequences included in the TCP packet.

VI. Identifying Trigger Candidates

In order to model a CPU design that will closely resemble that of a RISC-V based ECU and to inspect its behavior, the QEMU open-source CPU emulator [12] is utilized. QEMU not only allows for emulation of different architectures but also permits fine-grained inspection of values stored in the CPU registers. Our analysis focuses on the registers responsible for the handling of an arbitrary TCP packet.

A RISC-V based virtual machine (VM) running Debian Linux is emulated in QEMU. The operating system under test is Linux, as present-day TCUs are known to employ Unix-like operating systems [3]. Utilizing QEMU's capabilities, execution traces are generated. These traces include the CPU's registers and their values after each executed instruction. While tracing the VM's execution, we transmit a TCP packet from the host system to the emulated system. The transmitted TCP packet contains an identifiable payload that can be used to pinpoint CPU registers used for loading data bytes of the TCP packet.

Using the above method, the assembly code responsible for processing the main body of the TCP packet is extracted. The registers shown in Listing 1 are the ones whose values are influenced by the parsing of a TCP packet, regardless if it is accepted or rejected. Registers a4 and a6 hold the starting and ending memory addresses in between which the TCP data bytes are stored. Register a3 is used to hold the consecutive data bytes of the TCP packet. The provided assembly code loops until the last TCP data byte is processed and both a4 and a6 registers become equal.

The above examination revealed that the data from a TCP packet are continuously placed within the lower 32-bits of the a3 GPR. This information can be exploited by the attacker to place specific sequences of values in this part of the a3 GPR using a specially crafted TCP packet. Those sequences of values can then act as inputs to the triggering circuit to enable the generation of the triggering signal.

Listing 1: Assembly code run by the emulated 64-bit RISC-V CPU for the processing of arbitrary TCP packets.

```
Mnemonic        Operands
lw              a3 ,0(a4)
addw            a5 ,a5 ,a2
addi            a4 ,a4 ,4
addw            a5 ,a5 ,a3
sltu            a2 ,a5 ,a3
bgtu            a6 ,a4 ,-12
```

VII. Hardware Trojan Design

A HT is a purposeful modification by an attacker of a chip's original functionality to perform in an out-of-specification way. This induced behavior can be leveraged at the time of the chip's deployment by the attacker, to serve malicious purposes. The most common ways of implementing malicious circuitry modifications include either the modification of the chip's Register-Transfer-Level code at the design stage inside the design house or the direct modification of the chips' finalized layout at the fabrication stage inside the foundry. Of the two methods, the latter is the most difficult to implement but provides also the stealthiest results, as the induced modifications can only be traced through post-fabrication testing. Yet, testing for HT detection is considered to be very time consuming, expensive and even destructive in some cases [13].

Beside the insertion method of the HT, of major concern is the functionality of the HT itself. Depending on the HT's purpose, the attacker would need to devise an appropriate HT design that will evade unintended activation and provide a significant amount of attack controlability. Especially when it comes to the insertion of a HT inside an automobile, an attacker is not only interested in triggering the HT selectively for specific automobiles but also to be able to perform that at a large scale. For that reason the implemented triggering mechanism takes advantage of the wireless interfaces implemented on the TCU and the ability of the attacker to send specially crafted packets through them to the TCU hosting the HT. As for the HT's payload, it interferes with the state of the TCU's reset signal to keep the TCU in a constant reset mode and practically disable it. This in turn can lead to the immobilization of the autonomous vehicle/fleet of vehicles.

Our hardware trojan consists of two circuits, the triggering and the payload. The triggering circuit consists of a combination of synchronous and asynchronous logic, that monitors the sequence of bytes on a specific GPR, in order to raise the triggering signal. The payload circuit is simple and consists of only one MUX gate, that uses the triggering signal as the select signal to output either the correct value of the global reset signal or an "asserted" one that will disable the RISC-V CPU.

The triggering circuit consists of two parts, one implementing an asynchronous counter and the other implementing a reset logic for the counter. More specifically, the strategy described in [14] is utilized, in order to create a combinatorial logic that monitors the values loaded in GPR a3. This combinatorial logic consists of NAND and NOR gates, attached on the flip-flops of the a3 GPR, that create low transition probability signals. Three such low transition probability signals (*SEQ1*, *SEQ2*, *SEQ3* in Figure 1) are created, that take the logic value '1' only when a respective sequence value is loaded in GPR a3. For the generation of each signal 33 digital gates are required (15 AND, 16 OR and 2 NOT gates). The total

number of gates required for all three generated signals is 99 gates.

The generated sequence signals are fed to three distinct edge detectors (each consisting of 1 flip-flop, 1 AND and 1 NOT gate), as shown in Figure 1. The outputs of the three edge detectors pass through an OR and the outcome is sampled by a flip-flop. The output of this flip-flop is then fed in a 2-bit asynchronous edge counter, consisting of two flip-flops (blue dashed frame in Figure 1). The first flip flop takes as clock input the sampled signal of the edge detectors and has as input an inverted feedback of its output Q_{CNT1}. In turn this inverted feedback is connected to the clock input of the second flip-flop, while the input of the second flip-flop is again an inverted feedback of its output Q_{CNT2}. Both of the inverted feedbacks feeding the D_{CNT1} and D_{CNT2} inputs of the counter flip-flops, pass through a combinatorial logic (pink and yellow dashed frames in Figure 1). This ensures that the triggering sequences are only counted when seen in the correct order inside the a3 GPR. The two flip-flop outputs Q_{CNT1} and Q_{CNT2}, pass through an AND gate to produce the final triggering signal that will act as the select signal of the payload MUX gate. This asynchronous counter can count up to the binary value of '11', which equals the number of sequence values inserted inside the triggering TCP packet.

The above described logic is used to monitor the loading of three consecutive 4-byte sequences inside GPR a3. This provides our HT with a security of 96 bits. In the event of a HT detection testing, an evaluator would have to guess correctly the sequence values with a probability of $\frac{1}{2^{32}}$ for each sequence, giving a total probability of $\frac{1}{2^{96}}$ to guess all of them correctly and trigger the HT. Thus, this number of bits is high enough to be considered secure against intended activation attempts, as well as unintended activation events (e.g., random occurrences of the triggering sequences inside the monitored GPR). Furthermore, due to our HT's scalable design, it is possible for the attacker to increase the total number of sequence values necessary for the HT triggering. Each new sequence added requires the generation of an additional *SEQ#* signal, through the addition of 33 extra digital gates. Depending on the total number of sequences used, the flip-flops in the asynchronous counter should increase accordingly.

The second part of the triggering circuit consists of the reset mechanism for the flip-flops of the asynchronous counter. The reason behind the separate reset mechanism is twofold:

- The triggering sequence values might appear randomly outside of our triggering TCP packet as part of other executed programs. Thus, the counter should not consider these occurrences in the count.
- In addition, the Q_{CNT1} and Q_{CNT2} outputs should keep their values when the global reset is asserted, resetting all the rest of the memory elements.

As the malicious TCP packet is being processed by the CPU, the a3 GPR takes in succession the triggering sequence values as shown in the Assembly code in Listing 1. The flip-flop that is sampling the sequence signals, should sample a logic "0" value at most for one clock cycle (while the sequence values are interchanged in the a3 register and until they settle). This is expected as the added flip-flop might not be perfectly timed with the changes on the values of the a3 register, since the trojan is inserted at the fabrication stage.

The reset mechanism consists of two flip-flops and a combinatorial logic. The first flip-flop of the reset mechanism takes as inputs the OR of the *SEQ#* signals and acts as an edge detector, to alert for the start of a potential triggering sequence (*preReset* signal in Figure 1). If the sampled value remains at '0' for two or more consecutive clock cycles, then the triggering sequence is not being processed by the CPU and the two counter flip-flops should remain at reset. Thus, the *postReset* signal takes the value '1' and the *preReset* signal remains at '0'. These signals are then coupled with the global reset signal of the CPU (negative-logic global reset, constantly at logic value '1'), in order to produce the *Counter Reset* signal that keeps the counter's flip-flops at reset.

As described previously, the payload of the trojan is a simple MUX gate that has as inputs the CPU's original global reset signal and a modified one (a constant '0' as the global reset has negative polarity). The generated triggering signal serves as the select signal of the MUX. If the triggering signal is low, the MUX outputs the original CPU global reset to the CPUs modules. Once the triggering signal goes high, the MUX outputs the modified reset signal, thus resetting the CPU.

TABLE I: Gate count of trojan's circuits in ASIC implementations.

Number Of Digital Gates Per Module			
Module	***Logic Gates***	***FFs***	***Total***
Trigger Circuit	121	8	129
Payload Circuit	1	0	1
Total	122	8	130

The total number of digital gates needed for the implementation of the trojan is 130, with only 8 of them being flip-flops, as seen in Table I. The implemented mechanism allows for scalability as the triggering circuit can be adjusted to host larger or smaller sequences. This can be decided during the time of the HT insertion, according to the attacker's spatial or stealthiness needs. A larger sequence provides more run-time stealthiness as it makes it harder to trigger the trojan either accidentally or on purpose during testing. On the other hand, a smaller trojan facilitates insertion, as it requires less spatial resources on an ASIC layout and is harder to detect upon visual inspection of the finalized layout.

VIII. Hardware Trojan Implementation and Testing

We utilize the Genesys 2 FPGA board from Digilent to implement our HT on the Kintex-7 FPGA on board. Our RISC-V CPU target is Ariane [15], a modern, Linux capable 64-bit CPU design. Ariane's CPU clock runs at 50 MHz and boots a BusyBox implementation of the Linux kernel from a micro-SD card on board. The insertion of the HT is done by modifying Ariane's RTL code and the design is synthesized using Xilinx's

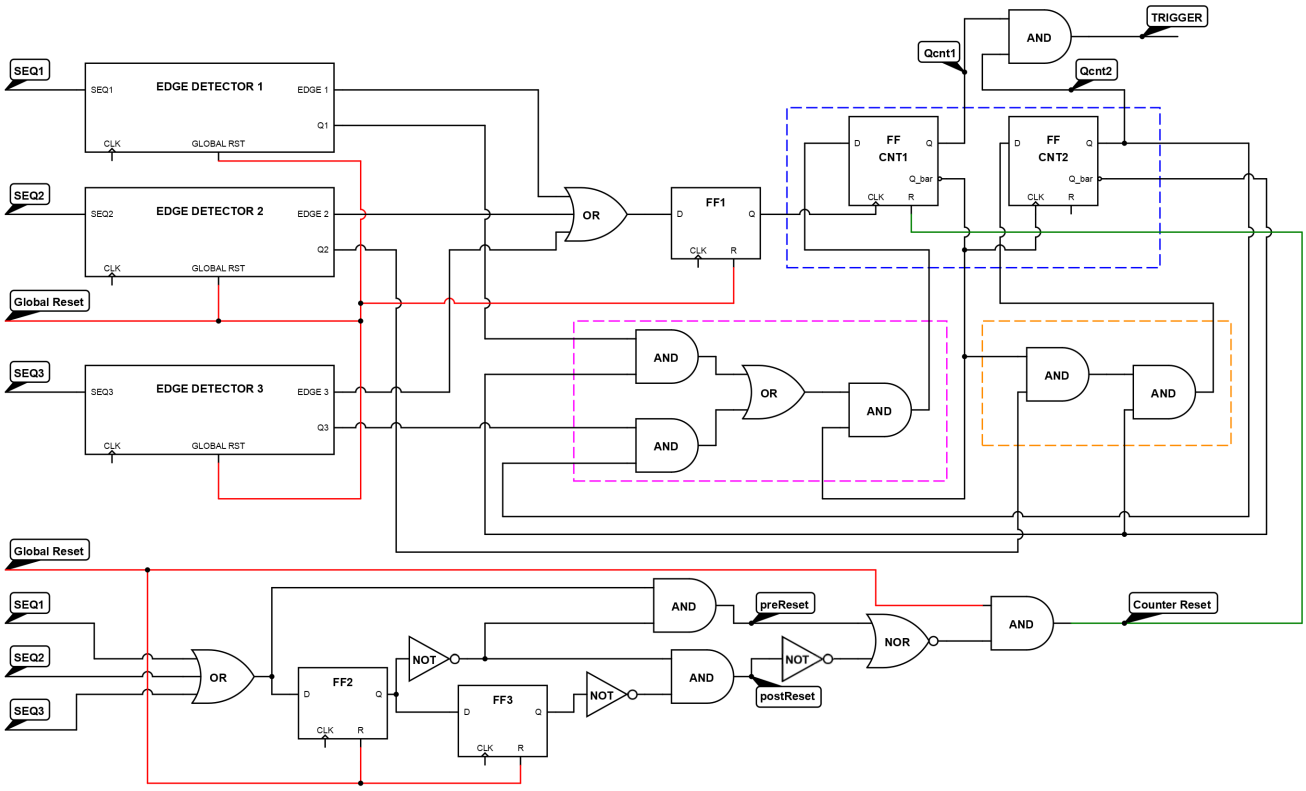

Fig. 1: Schematic of the triggering circuit.

Vivado tool. The FPGA utilization of the overall CPU design and the HT circuit can be seen in Table II.

TABLE II: Resource utilization of RISC-V CPU's and HT circuits' implementation.

Utilization Percentages In FPGA				
Module Name	***LUTs***	***FFs***	***LUT Utilization***	***FF Utilization***
Ariane[1]	66123	51875	32.4%	12.7%
Trigger Circuit[2]	32	8	0.00048%	0.00015%
Payload Circuit[2]	1	0	0.000015%	0%

[1]Utilization with respect to the total FPGA resources.
[2]Utilization with respect to Ariane's total implementation resources.

The low FPGA utilization numbers of the trojan circuit compared to the overall RISC-V CPU design is an indicator that the total gate overhead of our HT implementation can indeed be considered small compared to modern CPU designs. This can prove to be beneficial against methods that try to trace the existence of malicious trojan insertions in chips, using side-channel information [16].

To test the correct functionality of the HT, the code shown in Listing 2 is used in conjunction with an Integrated Logic Analyzer (ILA) core, to monitor the values of specific internal CPU signals. The code in Listing 2 integrates the TCP packet-processing assembly code (Listing 1) that was extracted using QEMU. The adjustments in the registers a7 (ILA core triggering) and a4/a6 (registers that store the memory locations of the TCP packet's data bytes) at the start of the code, do not alter the general principles encountered in a real scenario with a TCP packet being processed inside a RISC-V based ECU. Modified code versions of Listing 2 are also employed to validate that the HT is triggered only when the correct sequence of values appear in proper order inside GPR a3.

Listing 2: The user process testing the hardware trojan.

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main( ){
  u_int32_t test[3] = {X, Y, Z};

  asm ( "lui a7,0xfffff\n\t" \
        "sll a7,a7,0x14\n\t" \);
  asm ("mv a4, %0\n\t" :: "r"(test) : "a4");
  asm ("addi a6, a4, 0\n\t" \
       "addi a6, a6, 12\n\t" \);
  // Assembly that triggers trojan
  asm ( "loop:\n\t"
        "lw a3,0(a4)\n\t" \
        "addw a5,a5,a2\n\t" \
        "addi a4,a4,4\n\t" \
        "addw a5,a5,a3\n\t" \
        "sltu a2,a5,a3\n\t" \
        "bgtu a6,a4,loop\n\t" \);
  return 0;
}
```

IX. Discussion

A. Trigger Stability

Our triggering circuit relies on a specific GPR to store the TCP packet data, causing the HT's triggering mechanism to be vulnerable to changes in register allocation. Register allocations in the targeted software, the Linux kernel in this case, could change with updates to the software, compiler tool-chain, or even compiler options. The stability of register allocation is especially important due to the delay between the time when the HT is designed, to when it is integrated into a manufactured ECU, and finally when it is operational inside a vehicle. Additional work needs to be done to measure how stable controllable registers are over the natural evolution of software to better understand the real-world practicality of our proposed trigger design process.

B. Defenses

There are different approaches to limit the potential damage caused by HTs. On the automobile level, the resulting damage caused by the triggering of an HT can be limited through strict segmentation of the automobile's internal systems, regulation of their inter-communication and redundancy. Ideally a HT should be detected and stopped in its track through the means of an HT detection method before it is ever incorporated inside a vehicle. This requires validation at multiple points in the ECUs supply chain. Utilizing testing and design techniques such as those discussed by Xiao *et al.* can provide more confidence in the integrity of vehicle hardware [16].

X. Conclusions

Previous attacks against automotive systems required either physical access to ECUs of the target vehicle or a remotely exploitable vulnerability on the software running on one of the vehicle's ECUs. Our proof of concept implementation shows how a hardware trojan can be used to overcome the need for physical access during the attack stage, taking advantage of a vehicle's wireless connectivity interfaces. We demonstrate how knowledge of open-source software, run on CPU-alike ECUs, can be used to pinpoint remotely controllable registers in them.

Furthermore, this information is used to design a HT that monitors the values placed in such a register, awaiting for a triggering sequence provided by a malicious TCP packet. Once enabled, the payload of the trojan asserts the reset signal of the ECU to disable it. The insertion of such a HT in the TCU, which includes wireless interfaces with the outside world and is responsible for various vehicle's safety and communication features, can potentially lead to the immobilization of the vehicle, especially of an autonomous one.

References

[1] K. Koscher et al., "Experimental Security Analysis of a Modern Automobile," Jan. 2010, pp. 447–462. doi: 10.1109/SP.2010.34.

[2] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla From Wireless To CAN Bus," Black Hat USA, p. 16, 2017.

[3] S. Checkoway et al., "Comprehensive Experimental Analyses of Automotive Attack Surfaces," p. 16.

[4] Renesas, "RH850/U2B Zone/Domain and Vehicle Motion Microcontroller", Accessed June 21, 2022 [Online]. Available: https://www.renesas.com/sg/en/products/microcontrollers-microprocessors/rh850-automotive-mcus/rh850u2b-zonedomain-and-vehicle-motion-microcontroller

[5] NSI-TEXE, "RISC-V processor with vector extension certified for ISO 26262 ASIL D ready", Accessed June 21, 2022 [Online]. Available: https://www.nsitexe.com/en/ip-solutions/data-flow-processor/dr1000c/

[6] "RISC-V crypto core is qualified to ASIL-D for automotive designs ", Accessed June 21, 2022 [Online]. Available: https://www.eenewseurope.com/en/risc-v-crypto-core-is-qualified-to-asil-d-for-automotive-designs-2/

[7] M. Bozdal, M. Randa, M. Samie, and I. Jennions, "Hardware Trojan Enabled Denial of Service Attack on CAN Bus," Procedia Manufacturing, vol. 16, pp. 47–52, 2018, doi: 10.1016/j.promfg.2018.10.158.

[8] "Autonomous trucks lead the way", Accessed June 21, 2022 [Online]. Available: https://www2.deloitte.com/us/en/insights/focus/future-of-mobility/autonomous-trucks-lead-the-way.html

[9] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments," in IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009) , vol., no., pp.1-51, 15 July 2010, doi: 10.1109/IEEESTD.2010.5514475.

[10] "IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Networking Services," in IEEE Std 1609.3-2016 (Revision of IEEE Std 1609.3-2010) , vol., no., pp.1-160, 29 April 2016, doi: 10.1109/IEEESTD.2016.7458115.

[11] Jiang, R., Zhu, Y. (2019). "Wireless Access in Vehicular Environment." In: Shen, X., Lin, X., Zhang, K. (eds) Encyclopedia of Wireless Networks. Springer, Cham. https://doi.org/10.1007/978-3-319-32903-_309-1

[12] F. Bellard "QEMU, a fast and portable dynamic translator," 2005 Proceedings of the USENIX Annual Technical Conference, pp. 41–46.

[13] Sugawara, T., Suzuki, D., Fujii, R. et al. Reversing stealthy dopant-level circuits. J Cryptogr Eng 5, 85–94 (2015). https://doi.org/10.1007/s13389-015-0102-5

[14] Y. Su, H. Shen, R. Lu and Y. Ye, "A Stealthy Hardware Trojan Design and Corresponding Detection Method," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, pp. 1-6, doi: 10.1109/ISCAS51556.2021.9401770.

[15] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2629-2640, Nov. 2019, doi: 10.1109/TVLSI.2019.2926114.

[16] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor. 2016. Hardware Trojans: Lessons Learned after One Decade of Research. ACM Trans. Des. Autom. Electron. Syst. 22, 1, Article 6 (January 2017), 23 pages. https://doi.org/10.1145/2906147

Department of Defense, "About the Department of Defense (DOD)." Accessed April 18, 2017 [Online]. Available: https://www.defense.gov/About/