

# Disabling censorship on SPC5606B chips

Jan Van den Herrewegen  
Embercrypt BV  
jan@embercrypt.com

Faheem Adam  
Adam Engineering  
faheem@adamengineering.com

## Abstract

We propose an attack to disable the protection mechanism on a censored NXP SPC5606B, configured to use the public password. When censored, any flash memory read returns the same 16 byte block at address 0. Building on previous attacks on this chip (in the private password configuration), we uncover several anomalies when exposed to Voltage - and Electromagnetic Fault Injection (V-FI & EMFI), ultimately leading to the uncensoring of the device and giving us full read/write access to its flash memory. First, we analyse part of the bootloader code and direct our efforts to glitch that, gifting us a first plausible set of attack parameters. Though EMFI yields interesting results (e.g., the chip allows download with the private password), the flash memory remains censored. Finally, after establishing a power consumption side channel of a much earlier stage of the boot process, V-FI and EMFI pulses cause reads of the flash memory to return 16 byte blocks other than the starting block. Zooming in on this, we find the offset which disables censorship altogether, giving us full access to the devices flash memory. Finally, we discuss the plausible causes of the anomalous behaviour, and revisit the use and limits of fully automated approaches to fault injection.

## 1 Introduction

Many embedded chips provide a mechanism to restrict read and write access to the internal memory and registers in production. The NXP SPC5606B Microcontroller Unit (MCU) targets automotive applications and has several protection options, which are set and stored in an area of the shadow flash. The Boot Assist Module (BAM) is the code that handles the boot process in this MCU and other NXP MCUs. It samples dedicated external pins (Force Alternate Boot (FAB) and Alternate Boot Select (ABS)) upon boot to see if it should wait for an external bootloader to be transmitted from a debug tool over either UART or CAN. We will refer to this uploaded piece of code as the *secondary bootloader* from here on. The manufacturer can set a 64 bit private password to protect the JTAG debug interface and flash memory. Alongside, a public password (0xfeedface 0xcafebeef) is hardcoded in the BAM. Two shadow flash configuration words (Serial Sensorship Control (SC) and Censorship Control

(CW)) determine the exact security configuration. NXP describes this mechanism as *censorship* control, yielding four possible scenarios: 1. *Uncensored* - the default configuration which leaves the chip unprotected. 2. *Private password and uncensored* - flash memory is readable from any context (i.e., also from BAM code), thus when one provides the correct password, they can execute the secondary bootloader. 3. *Private password and censored* - From a black box perspective, the device behaves the same as in the *private uncensored* scenario (except for the fact that the words of the private password must be passed in the reverse order here). However, the flash memory is censored (i.e., inaccessible from the BAM). Thus, a separate module (the System Status and Configuration Module (SSCM)) must uncensor the device before control is handed over to the secondary bootloader. [O’F20] shows how to bypass this configuration. 4. *Public password and censored* - the datasheet describes this configuration as *chip lockout*, indicating there is no possible way of accessing or programming the chip. Likewise, any preliminary attacks on this configuration in [O’F20] did not yield any success. From here on, we will refer to this configuration as *chip lockout*. Here, the flash memory remains censored once the secondary bootloader executes.

Fault injection techniques rely upon injecting anomalies into the internal circuits of the chip, be it optical, voltage-based (Voltage Fault-Injection (V-FI)) or electromagnetic (Electromagnetic Fault Injection (EMFI)). EMFI is a technique commonly used to inject faults into embedded systems. A high voltage is built up over a coil positioned over the chips surface. Upon release, the electromagnetic emanation from the current flowing through the coil induces a current in the chip, causing bitflips and other faulty mayhem. Its relative affordability and ease to use (no target alterations required) make it an excellent mechanism to inject faults onto an embedded system. In the following experiments, we use a setup with a commercially available tool (ChipSHOUTER from NewAE Technology Inc., which costs around 3000 EUR), along with the common ARM-based Teensy microcontroller ( 50 EUR).

## 1.1 Previous work

In [O’F20], O’Flynn details an attack on various SPC56xx & MPC56xx series chips with the private password configured. For a more extensive overview of automotive firmware extraction and fault injection techniques, including V-FI and EMFI, we refer to [dH21]. In the more recent MPC57xx chips, the BAM has been replaced with Boot Assist Flash (BAF). The manufacturer can disable BAF activation using external pins. Wiersma et. al propose an attack on what seems to be such chips (referred to as ASILD2) in [WP17]. Their attack comprises of glitching the JTAG lock bits and life cycle encoding of the device.

To the best of our knowledge, this is the first published attack on the *chip lockout* configuration of the SPC5606B MCU. However, as O’Flynn correctly points out, various automotive tuners and diagnostic tool manufacturers (e.g.,

HPTuners <sup>1</sup>), already offer services for ECUs with this particular chip. Hence, they may already exploit the weakness we describe here. Only in scrutinising a security mechanism and understanding the ways we can compromise it, we can learn for the next generation of chips. It is in this train of thought we publish this paper.

**Contributions** In this paper, we show how to bypass *chip lockdown* on the SPC5606B by using EMFI.

- We scrutinise the censorship mechanism in the SPC5606B chip and expose the chip to various fault injection techniques. We believe the results of these experiments can help in better understanding the impacts of fault attacks and ultimately design more secure mechanisms.
- We pinpoint a critical section in the power consumption side channel to trigger an EMFI attack in order to bypass the *chip lockdown* configuration (i.e., public censored) of the SPC5606B.
- In support of reproducible research, we open source our code framework we developed for the EMFI attacks <sup>2</sup>.

**Responsible disclosure** We disclosed the findings presented in this paper to the NXP Product Security Incident Response Team (PSIRT) in October 2024. They did not independently reproduce our attack, but have no reason to refute the accuracy of our findings. They also note that the MPC56XX family of legacy products was not designed nor claimed to be resistant against EMI attacks or other physical attacks.

## 2 Attacking censorship

In this Section, we detail the attack on the SPC5606B configured in *chip lockdown*. We created a secondary bootloader which simply outputs the first 32 byte of the flash memory. We performed all experiments on a SPC5606B chip with a 16 MHz crystal. We communicate over the Universal Asynchronous Receiver-Transmitter (UART) interface with a baud rate of 19200. Since the device returns a valid 16 byte chunk of memory (i.e., that located at address 0) even when censored, we know that censorship is no hardware encryption mechanism, but rather placed between the memory bus and physical flash memory.

### 2.1 Boot process

Table 2.1 details the several configuration options. In this paper, we attack the *chip lockdown* configuration (e.g., SC && CW != 55aa). Device *Censorship* concerns the access to the on-chip flash memory. In case the chip is censored,

---

<sup>1</sup><https://www.hptuners.eu>

<sup>2</sup>[https://github.com/EmberCrypt/Teensy\\_ChipShouter](https://github.com/EmberCrypt/Teensy_ChipShouter)

the SSCM is responsible to uncensor the flash memory before proceeding. The BAM code does so by writing the provided password to the corresponding SSCM registers. To better understand the attack vectors, we describe the devices behaviour in the separate configurations.

**public censored** Since the password is public, we can always execute the secondary bootloader, however the flash memory remains censored. Only the SSCM can disable censorship by writing the private password to the appropriate SSCM registers. With censorship enabled, the chips flash memory appears as the same repetition of a 16 byte sequence (the first 16 byte in memory). Writes to the SSCM password register are only allowed in privileged context and thus are prohibited in the secondary bootloader.

**private uncensored** In this configuration, the device is uncensored, meaning that the BAM code can access flash memory. The provided password is compared to the password stored in shadow flash. Knowledge of this password is sufficient to uncensor the device.

**private censored** Here, only the SSCM can uncensor the flash memory and check the password. Thus, after the secondary bootloader download is completed, the SSCM writes the SSCM password registers, sets a timer and checks for success when the timer runs out. If the password is incorrect, the device halts.

SC \ CW		
	55aa (uncensored)	!55aa (censored)
55aa (private PW)	private uncensored	private censored
!55aa (public PW)	unprotected	chip lockout

## 2.2 BAM attack

In this section, we analyse a critical section of the BAM code and detail our attempts attacking it. Along with confirming the attacks on the private password described in [O’F20], we found several interesting anomalies, which we summarise here. Offsets are given from the start of the transmission of the last password byte (at a 19200 baud rate). Since a data byte (10 bits over the wire) transmitted over a 19200 8n1 baud connection takes  $520.83\mu s$ , the offsets given here fall roughly around  $8\mu s$  of the reception of the last byte.

### 2.2.1 Bootloader analysis

Figure 1 shows part of the bootROM (16kB mapped at FFFF\_C000) which determines the boot configuration and does the password check. As we can see from the code, the public password is hardcoded. One idea offered in [O’F20] to bypass the *chip lockout* scenario is to provide the correct private password and



2. **Glitch public**  $528.750\mu s$ ;  $409V$ ;  $1mmCCW$  : the BAM accepts the provided password and executes the bootloader but the flash memory is still censored. This behaviour is anomalous, since it seems the bootloader is still configured as *chip\_lockout* (e.g., the SSCM has not uncensored the flash memory), however allows the download with the private password. We speculate the BAM still takes the public password branch here, but the glitch affects one of the latter `se_bne` or `se_beq` instructions.

Regardless of the failure to disable censorship, an attacker would not know the private password in a realistic scenario and would still have to guess it somehow. Unless it has low entropy (e.g., all `ff` or `00`), this task is unfeasible over the slow UART or CAN connection. Moreover, a password including either `ffff` or `0000` is invalid and automatically rejected.

## 2.3 POR attack

In this section, we take a different approach and analyse a power consumption side channel to set up an EMFI attack.

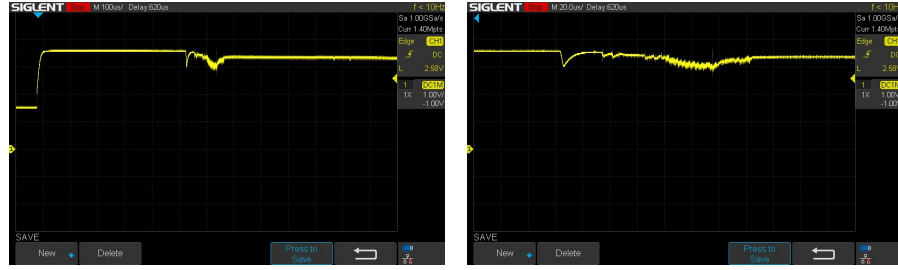
### 2.3.1 Power analysis

In many chips, a Power-On Reset (POR) pin invalidates all registers and forces the chip into a hard reset, as would be the case when power is first applied to the chip. Sadly, this is not the case on this chip, with the RESET pin only forcing a soft reset and thus not triggering any censorship queries. According to the datasheet, a POR is required before any changes in the shadow flash will take effect. Thus, we use the GIANt [Osw16], an open source voltage fault injection tool, to control the input voltage (on `Vdd`) directly - letting it drop below  $\sim 2V$  to force a POR and thus reload the censorship settings.

This enables us to analyse the fluctuations in core voltage upon boot, giving us an insight into its inner workings. As both SC, CW and the private password are stored in the shadow flash, we expect the loading of these registers to require a lot of power. We could obtain this side channel by placing a simple 50 Ohm shunt resistor between `Vss` and the ground of our power supply. To eliminate any external influences on our experiment, we desoldered the chip and connected only the reset pin, `Vdd` and `Vss`. Alternatively, since we do not connect any capacitors which mitigate voltage fluctuations, we can observe the voltage on the `Vdd` line upon reset on the oscilloscope, as shown in Figure 2. It shows a clear area of activity around  $500\mu s$  after applying the core voltage. This gives us first zone of interest to explore with fault attacks. While we could equally obtain this side channel by placing a simple H-Field Probe <sup>3</sup> over the chip and analysing its electromagnetic emanations, we gain the advantage here of leaving the full chip surface available, crucial for reliably injecting EMFI pulses.

---

<sup>3</sup>One we have used before is the NewAE CW505 Planar H-Field Probe



(a) Core voltage on power-on

(b) Core voltage during the critical section

Figure 2: Core voltage fluctuation on power-on and zoomed in on the critical section

**EMFI trigger** An EMFI attack requires a precise trigger point from which to initiate the glitch. We cannot trigger on the reset pin since this only performs a soft reset and does not reload censorship status. Furthermore, we tried triggering on the  $V_{dd}$  pin surpassing the brownout voltage detection (2.8V), but noticed the start offset of the critical section shown in 2 still varies too much for a precise trigger. Thus, we use the Analog-to-Digital Conversion (ADC) module on the commonly available Teensy 4.0<sup>4</sup> to detect when the area of high activity shown in 2 commences. We do this by triggering off the start of the critical section (occurring roughly at  $520\mu s$  after POR), characterised by the following sequence (with  $V_{dd} = 3.3V$ ): 1.  $V_{dd} = 0V$ : this holds the chip under reset. 2.  $V_{dd} > 2.8V$ : this initiates a POR, with this phase taking on average around  $500\mu s$ . 3. Steep drop in voltage ( $V_{dd} < 2.69V$ ).

We trigger after a certain low threshold has been reached on  $V_{dd}$ . This voltage is arbitrary since it depends on the nominal  $V_{dd}$ , but it ensures an accurate trigger for the critical section. Note that this trigger methodology relies on the core voltage fluctuating, which is only possible on a desoldered chip without any external capacitors to stabilise  $V_{dd}$ . Should the attack be reproduced on an ECU for instance, one could create a setup placing an H-Field Probe over the chips surface, as described in Section 2.3.1. The EMFI injection tips we used are the standard tips delivered with the NewAE Chipshouter (1mmCW, 1mmCCW), with only the winding direction differing.

**V-FI Chunk leakage**  $618 - 620.75\mu s$ ; *triggered from POR (V-FI)* Since we control  $V_{dd}$  anyway with the GIANt, we were all set to do initial scan of the attack surface by voltage glitching. This already yielded some promising results: the glitch makes the censorship mechanism return a different 16 byte chunk of memory. It seems that the SSCM is configured to take a memory chunk from arbitrary flash memory, instead of from address 0. We did not investigate further to understand how the SSCM selects which chunk to return. Though occurring plentifully (success rate of 20-40% in this offset range), we could not entirely

<sup>4</sup><https://www.pjrc.com/store/teensy40.html>

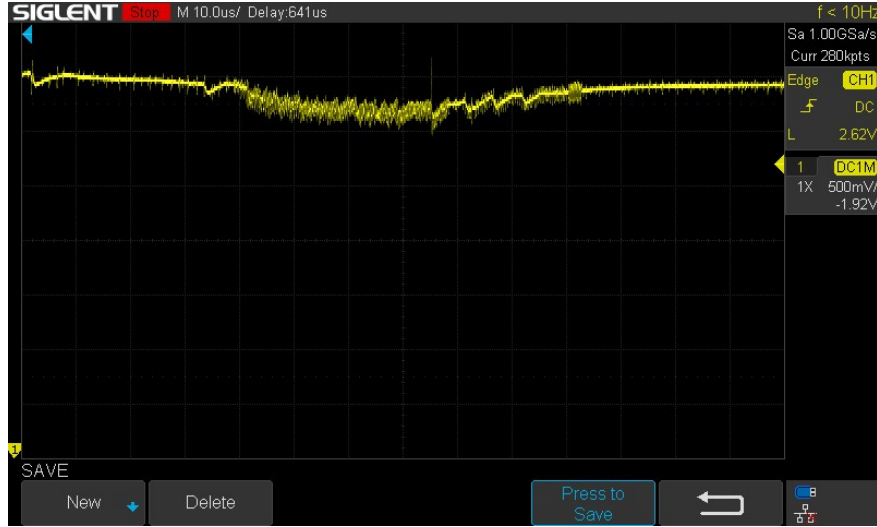


Figure 3: Successful glitch pulse at the end of the critical section (short pulse at  $75\mu s$  in this sample)

disable censorship, leaving us wanting.

**EMFI Chunk leakage**  $90\mu s$ ;  $460V$ ;  $1mmCCW$  When targeting this area with EMFI, we observe the same phenomenon. With success rates of up to 70%, we note that this attack occurs very frequently when correctly positioned on the chips surface, regardless of the injection tip, glitch voltage.

**Censorship disable**  $89\mu s$ ;  $300V$ ;  $1mmCW$  With these parameters we managed to prevent the chip enabling the censorship mechanism. Thus, we could simply download the bootloader using the public password and access the uncensored memory. We note that the success of the attack is very sensitive to the exact glitch parameters used, as well as the position of the injection tip, only occurring rarely. The glitch timing falls near the end of the critical section, as shown in Figure 3.

Once we can bypass the censorship mechanism we have full access to the flash memory. All that is left then is to execute a secondary bootloader which uploads new values of SC and CW to the shadow flash to uncensor the device.

### 3 Discussion

**Attack causes** Though it is difficult to figure out exactly what our EMFI attack glitches, we speculate it might be the setup of the SSCM. Since only an exact value (i.e., `55aa`) disables the censorship, while all other values enable it, it seems plausible that the device is by default configured as censored. When



subject to an EMFI pulse at  $90\mu s$ , the device changes which 16 byte chunk is returned upon a read of a censored area, indicating setup activity of the SSCM. The glitch disabling censorship falls  $1\mu s$  before this. Here, we discern two scenarios: 1. The glitch actually caused the read of CW to return `55aa`. This would automatically disable censorship on the device. Glitching a flash word to appear exactly as `55aa` seems unlikely, but only further EMFI experiments on this device could completely rule out this possibility. 2. To us, a more realistic scenario would be that the read of CW still returned the original value (i.e., `!55aa`), but the glitch caused a fault in the SSCM when setting up censorship, leaving the device uncensored.

**Parameter search** Many authors have attempted to devise algorithms to optimise searching the parameter space (e.g., [PWMM23, sha19]). With more intricate fault injection techniques (e.g., EMFI over V-FI), this problem is only more relevant. With this paper, we advocate for a multifaceted approach, comprising of both code analysis and side channel analysis and the use of several fault injection techniques altogether. In fact, after only observing partial results when glitching the BAM code, we needed to step back and widen the attack surface. Genetic algorithms that efficiently search the parameter space as proposed in [PWMM23, sha19] depend on anomalous outputs of the chip under attack. Often, an obvious offset to start experimenting with glitches (e.g., Figure 2) turns out to yield results quickly (as we noticed by voltage glitching the critical boot section). However, it seems to hit the physical limits of the fault injection technique (i.e., voltage fault injection being too inaccurate), inviting us to progress to a more intricate technique. Taking the first step with a more generic fault injection technique allows us to pinpoint an area of interest quickly, since successful faults with a more targeted approach (EMFI) occur far less frequently. We also note that a given injection tip (CCW), though also yielding promising results, is equally limited in completely disabling censorship, which we only attained with a clockwise-wound injection tip, and at a much lower voltage.

This all to show that, while a genetic algorithm certainly has its place in fault injection techniques, in our case a fully automated approach would not have sufficed. Once we hit the physical limits of the employed technique, we must step back, manually intervene and change out some hardware components if not resort to a more accurate injection technique. Therefore, we believe a hybrid algorithm which takes into account results from several injection techniques would be suitable here. We leave the design and implementation of this up to future research.

## 4 Conclusion

In this paper, we propose an EMFI attack to disable censorship on a NXP SPC5606B chip configured to use the public password, which is described as the chip lockout scenario in the datasheet. We explore attacks on a critical

code section of the BAM (Boot Access Module), the module which handles the boot process. Injecting EMFI pulses here cause the chip to accept the private password even when configured to use the public password. However, censorship remains enabled - causing the chip to return the same 16 byte chunk for any flash memory read. We then establish a power consumption side channel and identify a high activity section during the chip startup. Both voltage - and EM pulses at this offset make the censored device return a different 16 byte chunk of memory upon a read operation. Finally, an EMFI glitch falling just prior to this offset disables the censorship and gives us complete access to the chips memory. Our attack shows the importance of using a multi-faceted approach which performs a full scan of the attack surface with different fault injection mechanisms, narrowing down the parameter search space along the way.

## References

- [dH21] Jan Van den Herrewegen. *Automotive Firmware Extraction and Analysis Techniques*. Phd thesis, University of Birmingham, 2021. Available at <https://etheses.bham.ac.uk/id/eprint/11516/>.
- [O’F20] Colin O’Flynn. BAM BAM!! on reliability of EMFI for in-situ automotive ECU attacks. Cryptology ePrint Archive, Paper 2020/937, 2020.
- [Osw16] David Oswald. Generic implementation analysis toolkit. available online at <https://sourceforge.net/projects/giant>, 2016.
- [PWMM23] Enrico Pozzobon, Nils Weiß, Jürgen Mottok, and Vaclav Matousek. *Fuzzy fault injection attacks against secure automotive bootloaders*. Ruhr-Universität Bochum, 2023.
- [sha19] *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):199–224, Feb. 2019.
- [WP17] N. Wiersma and R. Pareja. Safety != security: On the resilience of asil-d certified microcontrollers against fault injection attacks. *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, page 9–16, Sep. 2017.