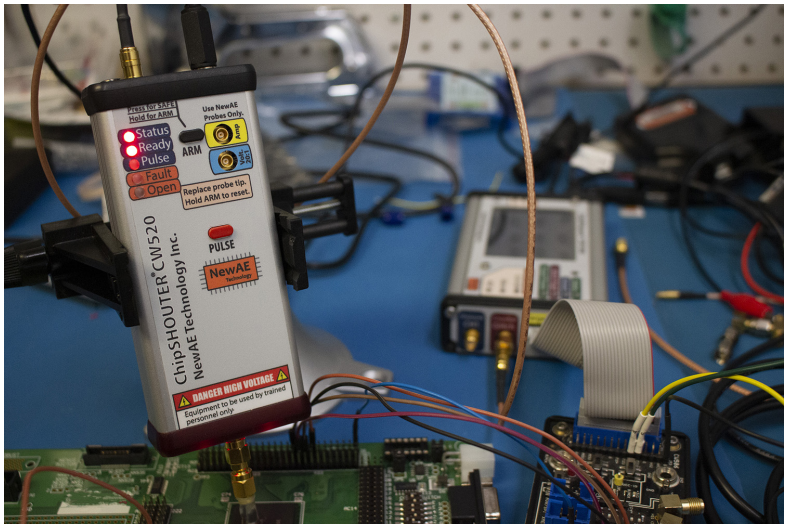
NAEAN0011: Electromagnetic Fault Injection (EMFI) for Automotive Safety & Security Testing with ChipSHOUTER®



Abstract: *Fault injection is the process of causing incorrect operation of a device, such as a CPU or memory element. From a safety design standpoint, this simulates random failures in the field, including many of the faults specified in ISO-26262:11-2018. From a security standpoint fault injection can be used to bypass authentication and security code. This paper discusses how electromagnetic fault injection (EMFI) tooling can be used for fault injection testing for both safety & security on an unmodified production ECU.*

Author: Colin O'Flynn
Last Update: Oct 19, 2020

Contents

1 Fault Injection

Fault injection implies that we are causing faults to occur in a device. A fault is generally when a device is performing an operation abnormally – so we can imagine all sorts of possible faults. But from a safety & security aspect, there is an interesting overlap of certain types of faults in digital devices.

Faults can also be called glitches – in the security research industry the process of performing *fault injection* is often called *glitching*. By changing characteristics of the fault injection method, we can achieve surprisingly reliable results.

With most common fault injection techniques, it is possible to cause a single instruction to 'skip' execution for example. It is often even possible to cause a certain number of bits to be set to 1 or reset to 0. With advanced fault injection techniques such as laser fault injection, we can target a single bit in a register even.

1.1 Fault Injection Tooling

Early work on fault-tolerant computing was done at NASA JPL, with proposals for the JPL STAR computer around 1961.[1] The idea of fault injection for testing these computers was published before 1972,[2] and the fact that memory could have the *information stored* damaged by strong fields was known from at least June 1970 (application date of U.S. Patent 4,413,327).[3] Inserting *internal* faults with external test tooling was known in at least the 1980s, with radiation-based fault injection used for testing error detection schemes.[4]

In the security area, the idea of fault injection is relatively newer, having been published since at least 1996, with A. Ross and M. Kuhn extensively demonstrating both clock and voltage glitches[5] to bypass security methods on devices. Let's briefly discuss the methods as used in the security area, which have been highly tuned for precise fault injection capabilities.

One of the original methods used for fault injection in the security area was clock glitching. Clock glitching involves inserting additional rising edges into the input clock of the device, with the objective of violating timing constraints

[1] Algirdas Avižienis. "Design of fault-tolerant computers". In: *Proceedings of the November 14-16, 1967, fall joint computer conference*. AFIPS '67 (Fall). New York, NY, USA: Association for Computing Machinery, Nov. 1967, pp. 733–743. ISBN: 978-1-4503-7896-3. DOI: 10.1145/1465611.1465708. URL: https://doi.org/10.1145/1465611.1465708.

[2] A Avizienis and D.A. Rennels. "Fault-tolerance experiments with the JPL STAR computer." In: San Francisco, CA, Jan. 1972.

[3] Joseph D. Sabo and Joel A. Karp. "Radiation circumvention technique". Pat. US4413327A. Nov. 1983. URL: https://patents.google.com/patent/US4413327A/en.

[4] U. Gunneflo, J. Karlsson, and J. Torin. "Evaluation of error detection schemes using fault injection by heavy-ion radiation". In: *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*. June 1989, pp. 340–347. DOI: 10.1109/FTCS.1989.105590.

[5] Ross Anderson and Markus Kuhn. "Tamper resistance: a cautionary note". In: *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*. WOEC'96. USA: USENIX Association, Nov. 1996, p. 1.

in the target device. For this to function, the clock must be used directly by the internal core. This means clock glitching will not be effective against two large classes of devices: those using internal oscillators, and those that use a Phase Lock Loop (PLL) to derive a new clock from the external clock[6]. The majority of high-performance devices fall into the latter category, as they will run the internal core at a much higher frequency than the external clock.

With voltage fault injection, a narrow "glitch" is inserted into the power supply of the digital device. This glitch is typically so narrow that brown-out detector and reset circuitry will not be triggered by it. An example of such a glitch is shown in Figure 1. Interested readers are referred to the footnote for more details of the capabilities of this specific fault module.[7]

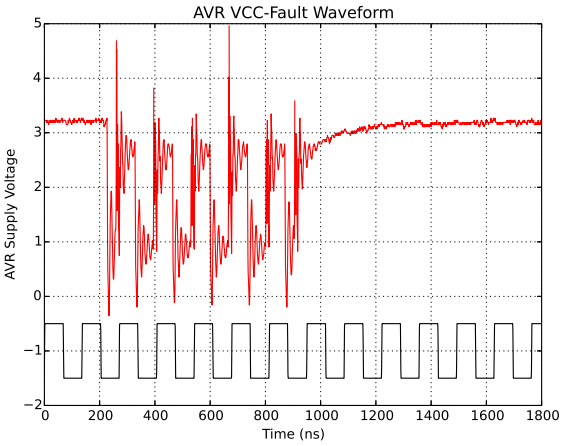


Figure 1: Very narrow glitches may not trigger brown-out detectors, while still causing faults.

Fundamentally, such glitches actually insert ringing into internal voltage nodes, and do not actually need to transfer the full "glitch depth" to the internal nodes.[8] While there are multiple underlying physical causes of the glitch, the most prominent appears to be the introduction of timing violations. These timing violations will corrupt data as it is loaded, stored, or used. This 'data' can include program instructions for example, beyond just main memory. Note that voltage glitching *can* corrupt data at rest, and not only data while it is

[6] Some clock glitches can be inserted 'through' the PLL by forcing the PLL to output invalid frequencies, but with less precise temporal positioning than if the chip uses the clock directly.

[7] Colin O'Flynn. *Fault Injection using Crowbars on Embedded Systems*. Tech. rep. 810. 2016. URL: http://eprint.iacr.org/2016/810.

[8] Loic Zussa et al. "Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter". In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. May 2014, pp. 130–135. DOI: 10.1109/HST.2014.6855583.

being moved.[9] Thus ringing may be the prominent mechanism, but this does not mean the only such one.

More advanced methods of performing fault injection tooling include electromagnetic fault injection (EMFI) and laser fault injection (LFI). Both have the advantage of allowing the location of the fault to be scanned over the surface of the target device. LFI will allow the most precise position - it's possible to flip single bits in a register with LFI for example. The main downside of LFI is the cost of the equipment, which includes not only the laser station, but positioning tables, target preperation equipment, etc.

We will discuss EMFI in more detail in Section 3.

1.2 Security

Fault injection is a powerful attack because our security code typically includes something like Listing 1 – in this example checking a firmware image has passed validation tests.

Looking at an assembly version of this in Listing 2 should make it clear how easily this test could be bypassed if we had the ability to skip instructions (i.e., cause them not to be executed) or corrupt values. Both of these would allow us to break the security assumptions. Bypassing instructions works because, for example, we could skip the branch instruction. We could also skip the load of `r0` with the comparison value (`1`), resulting in the function call return value compared with a previously loaded value of `r0` (in this case it was previously loaded with `0`).

Corrupting values in memory could allow us to change return values or flags. This can be powerful when specific "magic values" are being used, and if we corrupt those magic values the system may go into a less secure failsafe mode. This was used in bypassing the LPC series microcontroller read-out protection.[10]

```
if (check() == 1){
    load_firmware();
} else {
    panic();
}
```

Listing 1: Example security check code in C.

This is just the most basic form of fault injection. Other fault injection techniques can be used to break cryptography – for example a simple attack on RSA allows you to recover the RSA private key from a signing operation by

[9] O'Flynn, *Fault Injection using Crowbars on Embedded Systems*.

[10] Chris Gerlinsky. "Breaking Code Read Protection on the NXP LPC-family Microcontrollers". In: 2017. URL: https://recon.cx/2017/brussels/resources/slides/RECON-BRX-2017-Breaking_CRP_on_NXP_LPC_Microcontrollers_slides.pdf.

```
    ldi r0, 0
    ldi r0, 1
    rcall check
    cmp r4, r0
    bne fail
    rcall load_image
    ret
fail:
    rcall panic
    ret
```

Listing 2: Example security check code compiled into assembly.

injecting a *single fault almost anywhere in the computation.*[11] Fault attacks are known for most popular cryptographic algorithms.

1.3 Accelerated Safety Testing

Fault injection in safety critical systems is typically used as a testing tool in order to validate safety-critical code. Such fault injection was typically achieved with more deterministic methods such as special "fault injection" test harnesses, standard debug connections and emulators which can be used to change program flow or flip bits, and special purpose fault injection simulators.

We can consider the case of accelerating safety testing by instead optimizing our faults for specific objectives, rather than randomly flipping bits in memory as we might expect to occur in the field. This objective-based fault injection more closely mirrors security testing, where the fault injection is performed in order to bypass specific code, or corrupt specific values.

We will now look briefly at some of the fault models introduced in ISO 26262 that are relevant to faults in digital devices.

[11] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults". en. In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by Walter Fumy. LNCS. Berlin, Heidelberg: Springer, 1997, pp. 37–51. ISBN: 978-3-540-69053-5. DOI: 10.1007/3-540-69053-0.

2 ISO 26262 Fault Injection Testing

As part of ISO 26262, fault injection testing is discussed in several sections. Of most interest to us is ISO 26262-5:2018 (*Product development at the hardware level*), ISO 26262-6:2018 (*Product development at the software level*), and ISO 26262-11:2018 (*Guidelines on application of ISO 26262 to semiconductors*).

2.1 Types of Faults

Fault injection at the semiconductor component level is widely discussed in ISO 26262-11:2018, especially in clause 4.8.1 of ISO 26262-11:2018. Many of the methods rely on special instrumentation added to the device, or tooling which is difficult to use in a standard development lab (such as a particle accelerator, which does not yet fit on your desktop). Briefly, the events inserted are often characterized into some of the following types:

- Single Event Transient (SET): momentary voltage excursion (e.g. a voltage spike) at a node in an integrated circuit caused by the passage of a single energetic particle.
- Single Event Upset (SEU): A soft error caused by the signal induced by the passage of a single energetic particle;
- Single Bit Upset (SBU): A single storage location upset from a single event.
- Multi Cell Upset (MCU): A single event that induces several bits in an IC to fail at the same time. The error bits are usually, but not always, physically adjacent
- Multi Bit Upset (MBU): Two or more single-event-induced bit errors occurring in the same nibble, byte, or word. An MBU may not be corrected by a simple ECC (e.g. a single-bit error correction).

Note that a SET (a voltage spike) at a node can become a SEU depending on the location of the SET. If the voltage spike causes a register cell bit to flip, it now becomes a SEU or SBU. Thus we can consider that inserting a SET, with some ability to control location, should allow us to insert a wide variety of other faults.

2.2 Effects of Faults

If we consider the various types of fault enumerated, the effect of them will also vary depending on the type of device. Assuming our primary interest is digital devices, we can use Table 30 of ISO 26262-11:2018 which enumerates examples of various faults for different types of devices.

Looking at a microcontroller specifically, we can see four types of faults enumerated in Table 30 of ISO 26262-11:1018 for a CPU device are:

- `CPU_FM1`: given instruction flow(s) not executed (total omission).
- `CPU_FM2`: un-intended instruction(s) flow executed (commission).
- `CPU_FM3`: incorrect instruction flow timing (too early/late).
- `CPU_FM4`: incorrect instruction flow result.

In general, we can see that a SET (voltage spike) inserted at a localized area would cause timing of the registers to shift. The SET could also cause incorrect levels at the input or output of a register, causing an invalid value to be loaded. We will shortly demonstrate how all of these fault effects can be caused with an EMFI tool in a "desktop lab" environment.

The EMFI tooling is especially useful as it does not require modification to the target device. In fact, it's possible to perform the majority of this testing on production devices without any special debug or other interface required. This will be demonstrated for safety testing in Section 4, and for security testing in Section 5.

3 Electromagnetic Fault Injection

The objective of electromagnetic fault injection is to ultimately inject a voltage onto the structure of the die itself – this can cause both "soft-error" faults (such as bit flips in a register or SRAM), or temporary errors in reading voltage levels (a SET). The fact that a strong field has the ability to corrupt data (without damaging the device) has been known since at least 1970,[12] as previously mentioned.

For EMFI, the fault injection method is a rapidly changing magnetic field – so what we need is a tool to generate the changing field. The device that generates this localized field is the NewAE Technology Inc. ChipSHOUTER in this whitepaper (available as part number `NAE-CW520-KIT`).

3.1 Injection Tip (Coil)

The "business end" of these tools use some form of a coil in combination with a high permeability material, normally a ferrite. This ferrite is designed to concentrate the magnetic flux in a smaller area, making it possible we could flip bits in part of the memory without crashing the entire device.

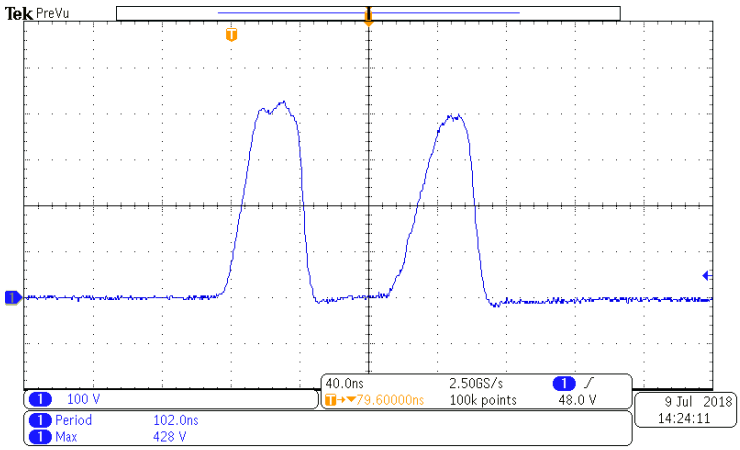


Figure 2: The drive waveform at the coil while inserting two pulses with a ChipSHOUTER.

A typical drive waveform for these tips is given in Figure 2. You'll notice the coil is driven to high voltages (400V) over a short period of time[13]. This is done by using a capacitor bank which is connected across the coil. Driving this

[12] Sabo and Karp, "Radiation circumvention technique".

[13] Our end goal is to generate a voltage in the device under test, which means we need a fast-changing magnetic field. Such a field is generated by a rapidly changing current in the injection coil. The high voltage is required as the inductive characteristics of the coil will 'resist' such a rapid change of current, forcing us to use a higher voltage to accomplish our desired current through the coil

coil requires a high-side switch to avoid keeping the output coil "hot" all the time for safety reasons.

3.2 ChipSHOUTER Tips

There are four injection tips included with the ChipSHOUTER. Two 4mm tips and two 1mm tips, each with both "clockwise" (CW) and "counter-clockwise" (CCW) versions. The size of the tips refers to the diameter of the ferrite core inside the coil, and the CW/CCW refers to the wrap direction of the wire around the ferrite core. The wrapping direction can be used to specify a positive or negative voltage induced into a specific target.

Figure 3 shows the outside of a coil, along with a clear version where you can see the wrap around the ferrite core. The variation between the various coils is shown in Figure 4.

Figure 3: Showing the ChipSHOUTER injection tip (coil).

Figure 4: Showing the wire wrapping direction around the injection tip (coil).

3.3 Ballistic Gel Target

The ChipSHOUTER kit (`NAE-CW520-KIT`) contains a target called the Ballistic Gel. This target has a large 32 Mbit SRAM chip, which can be used to under-

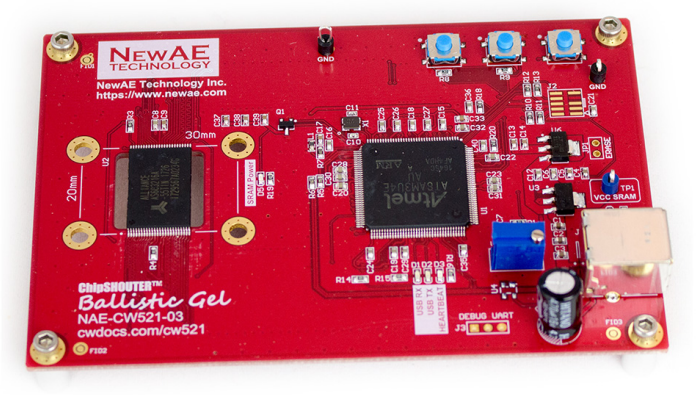


Figure 5: The Ballistic Gel is a SRAM based target to understand how EMFI can influence SRAM.

stand how bit flips vary with different settings. By downloading a pattern to the device, injecting a fault, and seeing where the pattern flipped bits we can get an idea of the sort of effects that are possible. The physical board is shown in Figure 5.

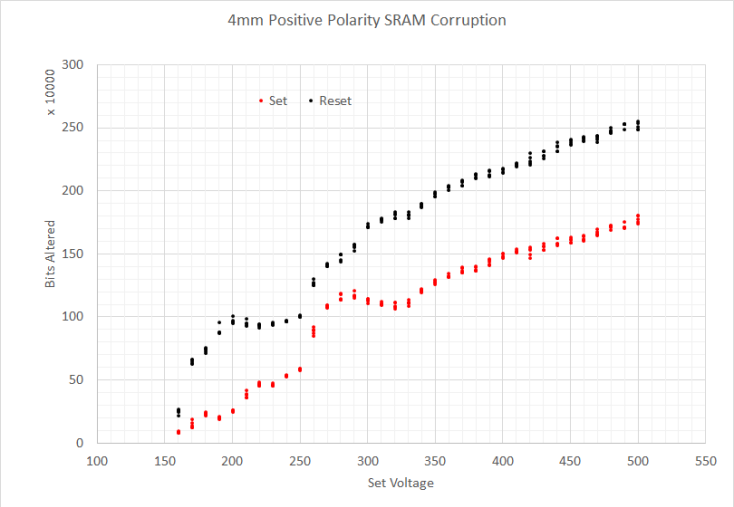


Figure 6: Jupyter CPA Table

From the ChipSHOUTER user manual, several examples of EMFI results have been given that we will duplicate here.

Figure 6 shows a graph of voltages the capacitor was charged to for the pulse compared to number of bits that flipped after injecting the fault. You'll notice that a higher voltage causes more bits to flip compared to a lower voltage. We've

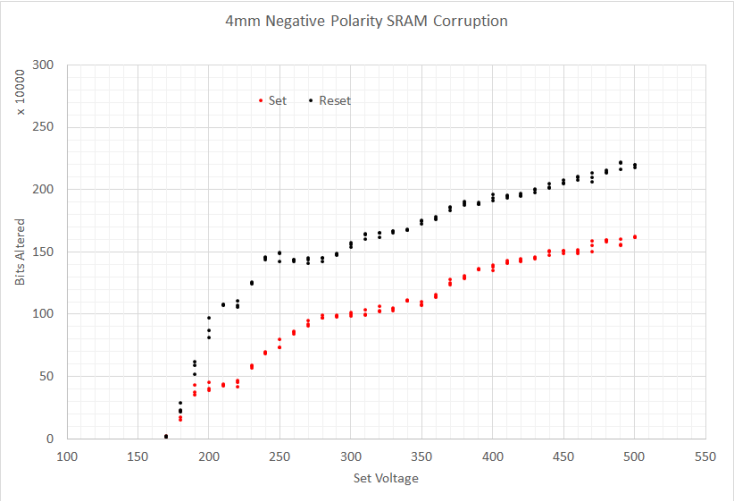


Figure 7: Jupyter CPA Table

also mapped bits that flipped from 1 to 0 (a bit reset) and flipped from 0 to 1 (a bit set).

We can compare the effect of the coil winding direction (clockwise vs. counter-clockwise) by looking at Figure 6. This is the same coil, but with the opposite winding direction. You'll notice there was a noticeable shift in the number of bit resets, presumably the opposite polarity voltage is being injected into the chip structure (that is - the transient voltage on the chip nodes).

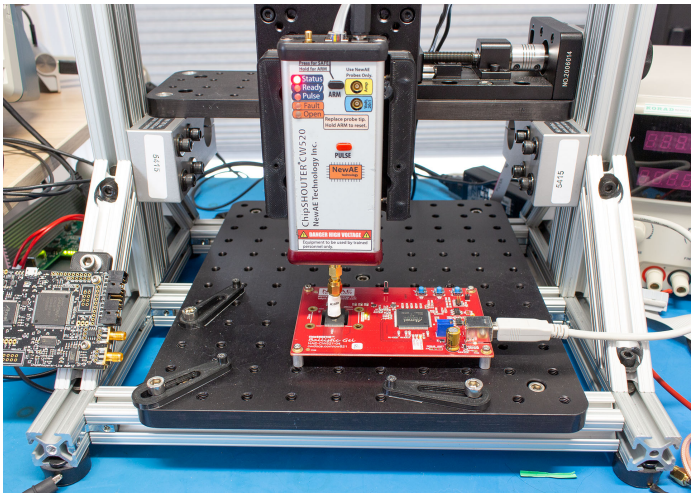


Figure 8: Ballistic Gel mounted on XYZ table with ChipSHOUTER.

3.3.1 Smaller Number of Bit Flips

In the previous examples, it was demonstrated how a large number of bits can be flipped using this target. Of course it may be interesting to target a much smaller number of events.

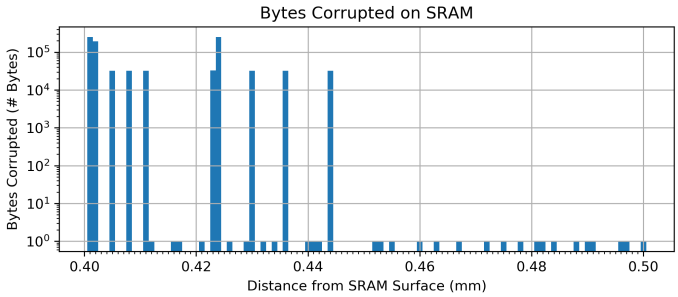


Figure 9: Changing the distance from the SRAM surface causes a differing number of faults to occur.

Such higher precision testing can be accomplished in two ways: the most flexible is the use of a precision XYZ table, such as the ChipShover. A photo of a prototype version of this is shown in Figure 8, where the ChipSHOUTER is mounted above the Ballistic Gel target.

We can see the affect of keeping the charge voltage constant (in this case 200V), but moving the tip away from the surface of the SRAM chip in Figure 9. Note that at some points a single byte is erroneously corrupted. Here a single attempt is made at each height, and thus there is some clear stochastic nature to the corruption occurring or not.

The sudden jump to a large number of errors is primarily explained by the structure of the SRAM chip itself. If the fault occurs on a path that affects multiple cells (such as a wordline or similar), it would be expected to cause the type of corruption seen here.

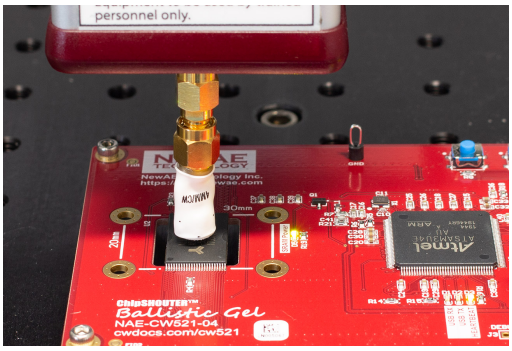


Figure 10: Close-up of ChipSHOUTER over the SRAM chip.

3.3.2 XY Scanning

As seen in the close-up of Figure 10, the ChipSHOUTER tip can scan over an area of the chip. Not only does this affect where the faults are injected, it can affect the number. As a simple example, scanning our same fault over the SRAM top gives us the results from Figure 11. Note that multiple areas have a 1-byte fault, some areas have more than 1-byte fault but not the large number of faults seen previously, and other areas have the large number of faults.

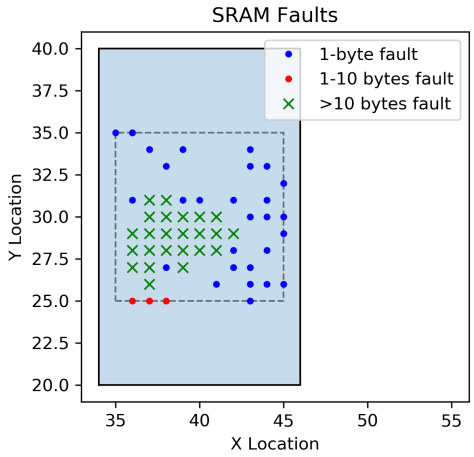


Figure 11: SRAM Fault Results (200V), light blue is SRAM chip top, dashed area shows scan zone.

3.4 More Details on ChipSHOUTER

You can find more details at https://www.newae.com/chipshouter for the product page. The ChipSHOUTER has associated open-source Python API and other tools posted at https://www.github.com/newaetech/chipshouter.

4 Production ECU Safety Example

This will demonstrate how the ChipSHOUTER can be used to perform a no-knowledge validation of an ECU to determine if corrupt settings or memory could cause safety flaws. In this example, an ECU from a production vehicle will be used such that the device is as close to "real life" as possible.

4.1 Target System

In this case, the example target will be an ECU from a 2006 model year vehicle. This relatively old ECU was chosen due to some existing public work discussing the potential for software failures to occur without tripping the expected fail-safe behaviour.[14]

We will in particular be interested in behaviour of the control loop for the throttle body. The following safety testing is not performed on a full system – that is, the safety testing is not showing overall system failure, as other failsafes are not investigated in this example.

Instead, we are simply demonstrating how EMFI can be used to trigger failure modes, without *any* knowledge of the system under test. More advanced analysis can be performed if the tester does have knowledge of the system under test, for example it is possible to compare memory and register dumps in order to understand where corruption is being inserted.

4.2 Test Bench

In order to perform a safety check, we need to operate the system inside of some normal bounds. In this case, a very simplistic engine/car simulator is built around the ECU to allow the system to operate normally.

This test bench is shown in Figure 12, which contains the following items:

1. The main ECU board we are testing.
2. The physical throttle body, wired to the ECU as normal.
3. The accelerator pedal sensor, wired to the ECU as normal.
4. An ignition switch (switches power on/off as a key would).
5. A simulator to provide cam & crank signals to the ECU.
6. A standard diagnostics reader to monitor the ECU datastream.
7. Probes to monitor the signal driving the throttle body.
8. A ChipSHOUTER to provide EMFI tooling.
9. A XYZ stage to allow scanning of the ChipSHOUTER.

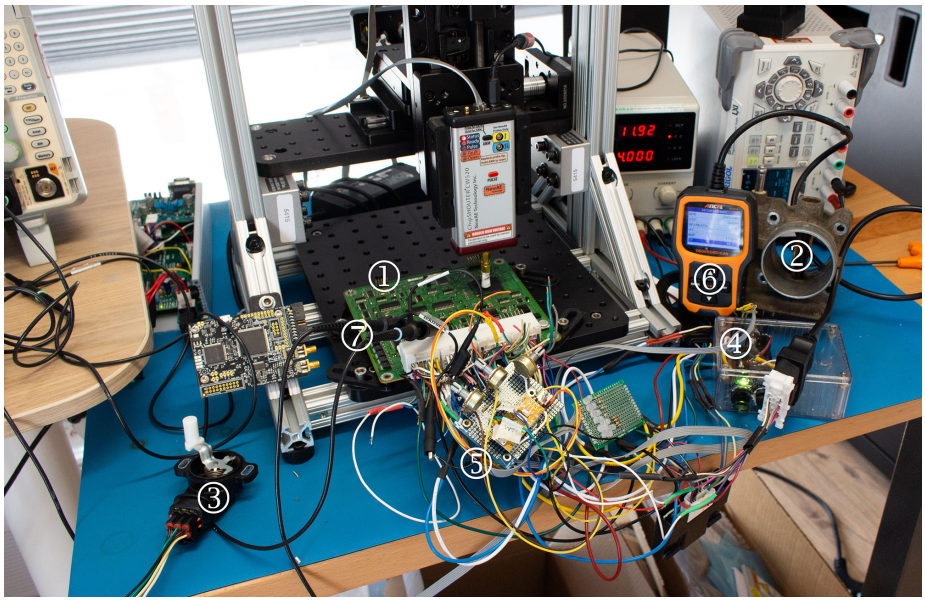


Figure 12: The test bench showing: ① the ECU under test, ② the throttle body, ③ the position sensor, ④ the ignition switch, ⑤ sensor simulator, ⑥ OBD-II reader, and ⑦ scope probes on PWM signal. A ChipSHOUTER EMFI tool is shown over-top of the main ECU.

The current setup does not automatically monitor the outputs, but uses a human-in-the-loop to observe out of specification modes (such as the throttle position not matching the commanded position).

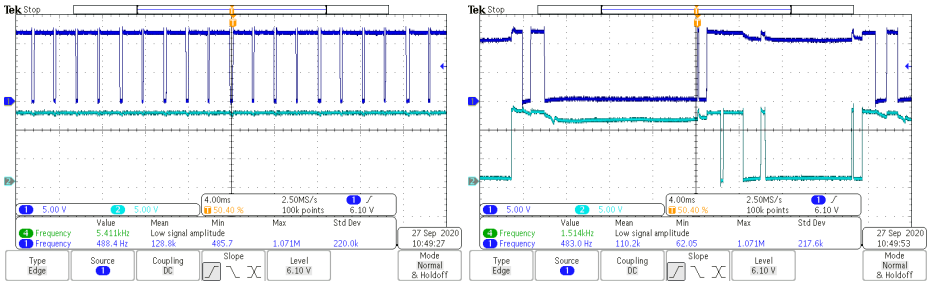
4.3 EMFI Results

While operating the device while performing EMFI, several failure modes were noted, including:

1. ECU resets and continues to operate.
2. ECU enters a fail-safe mode, such as reduced throttle opening.
3. The drive signals for the throttle body motor stopped operating normally.

The final failure mode is of the most interest, as it appeared the control loop was otherwise closed (the position followed the pedal in general). The motor is driven with pulse width modulation (PWM) signals, with a comparison of the two PWM modes shown in Figure 13.

[14] Phil Koopman. *A Case Study of Toyota Unintended Acceleration and Software Safety*. en. 2014. URL: https://ptolemy.berkeley.edu/projects/chess/pubs/1081/koopman14_toyota_ua_slides.pdf.



(a) Normal PWM Operation (b) Chattery PWM Operation

Figure 13: PWM output for throttle body control before and after EMFI insertion.

While the throttle body appeared to be maintaining the requested position, the throttle body now had a noticeable “chatter”. In addition, the power consumption jumped from the normal 1.6A to 3 – 5A (the current draw became less constant). It is assumed this was linked to the incorrect PWM waveform.

4.3.1 Stuck Throttle Results

Once in the incorrect PWM mode, the throttle would eventually stick either fully closed or fully open, and the accelerator pedal sensor changes no longer have any impact on the throttle position. In this case the current draw increased further, and viewing the PWM waveforms it was clear the output signal was now constant. Once this mode was entered, a power cycle was required to exit this mode.

While in this mode, the ECU continued to provide ignition output signals that responded to changes in the cam & crank signals, and the OBD-II scan tool could continue to provide diagnostics information. A photo of the throttle stuck open is shown in Figure 14. Note the throttle is shown commanded to 88% opening here. During regular operation, the maximum throttle opening on the test bench is only 81%, thus the 88% throttle position commanded here appears to be even beyond regular operating values.

4.4 Monitoring Data Corruption

In this case, the ECU debug port is locked. Thus we cannot easily monitor the corruption during operation to map the specific failures that are occurring. For example, we could be corrupting a CPU peripheral register, rather than the main memory. If you have full access to the debug information and memory usage of the device, you can perform analysis of the memory corruption to understand the actual safety mechanism being tripped (or not).

Figure 14: The ECU after an EMFI based fault caused some error where the throttle is commanded to become fully open.

4.5 In Vehicle Testing

In order to perform more complex safety testing, we can also demonstrate how this can be moved to an in-vehicle test. Figure 15 shows the ChipSHOUTER mounted on a much smaller gantry, which allows performing the positioning task in a smaller area. Such testing would allow more complete testing of multiple systems, in order to validate multiple fail-safes.

Figure 15: The ChipSHOUTER design makes it even possible to perform testing in-vehicle, without requiring extensive effort in the instrumentation process.

5 Production ECU Security Example

Note that the following section is a shortened version of the full work available from https://eprint.iacr.org/2020/937.pdf, which was presented at ESCAR-EU 2020.

The following security analysis work uses an ECU from a 2019 model year passenger vehicle. We will use EMFI to precisely bypass specific instructions. This will show how fault injection using EMFI is a powerful security analysis tool.

Note that the ECU we will demonstrate the security bypass on has publicly been 'tuned' from at least 2018[15], meaning that this (or a different) bypass has already been used in the field. This whitepaper is designed to help readers understand one method such car tuners could be using. Of importance is understanding how to perform this analysis yourself early in the design cycle, and not wait until products are in the field to discover such vulnerabilities.

5.1 Bootloader Password Details

This example will show how it is possible to bypass the password in the MPC55xx and MPC56xx series of microcontrollers, which allows full control of the ECU. Most devices in the MPC55xx and MPC56xx series include the Boot Assist Module (BAM) code which includes a serial or CAN bootloader mode. In these devices, various pin strappings are used to enable the bootloader. There is no method of disabling the bootloader entry, as the pin strapping takes priority over internal flash. Later devices in the MPC57xx series allow ignoring of external pin strapping.

The MPC5676R and MPC5566 devices (along with most others in the series) allow both a UART (serial) or CAN-based bootloader. Both of these bootloaders have a similar protocol, shown in Algorithm 1.

Algorithm 1: Boot Assist Module (BAM) as implemented by NXP

Result: Boots from SRAM
Receive an 8-byte password;
if *Password is incorrect* **then**
  STOP and wait for watchdog reset;
end
Receive 8 bytes of header information, including number of bytes (N) to follow;
Receive N bytes of bootloader program which will be written to SRAM;
Jump to SRAM bootloader;

In order to ensure data was received correctly, any processed data is echoed back. When using the UART bootloader, data is echoed back one byte at a time.

[15] See https://www.hptuners.com/product/l5p-ecm-exchange-service/

We can also see the difference in code execution by observing the correct and incorrect password processing, as in Figure 16. Here we measure time in cycles of the 20 MHz clock fed into the MPC5676R. Around 150 cycles before the end of the echo time (cycle -150 on Figure 16) we see the divergence of the power trace. Both of the last bytes for the correct and incorrect password was `0xEF` in this case, meaning the difference is related to code-flow only and not the data transferred. The assumption in this work is that inserting a code-flow change will allow bypassing of the security check.

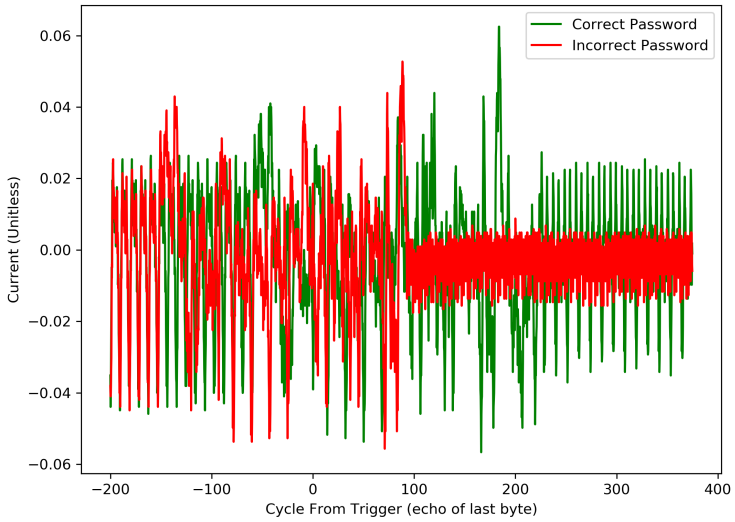


Figure 16: Comparing the correct and incorrect password makes the injection timing clear.

5.2 Password Attack

The attack on the BAM module assumes first that we have external control of the `BOOTCFG1` pin, which will force entry into the bootloader. From here the bootloader will perform Algorithm 1. With the device configured with a private flash password, there are two possible attacks setups:

1. Sending the public password `FEED_FACE_CAFE_BEEF`.
2. Sending an incorrect password (here we use `1122_3344_5566_7788`).

After each password is sent, a fault injection attack is attempted. There are six potential results from the attack:

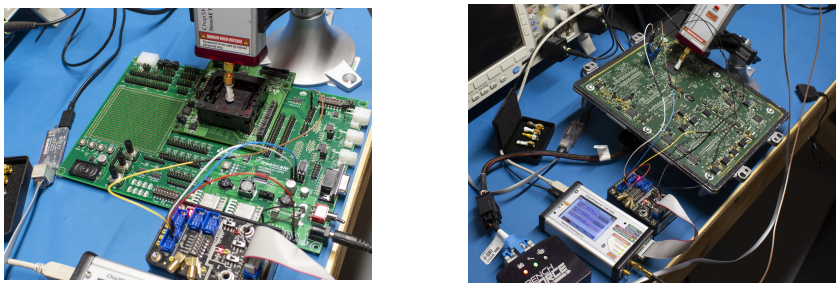
1. No effect (password not accepted): *Normal*.
2. Target reset: *Err-Reset*.
3. Password accepted, error during code download: *Err-Protocol*.
4. Password accepted, code downloads, but does not start: *Err-RunFail*.
5. Code downloads and runs, but flash access is still disabled: *Err-RunFail*.
6. Code downloads and runs, printing private (flash) password: *Success*.

In practice differentiating between (4) and (5) is unreliable – the default attack script appeared to access flash early on, and if the device was still disabled this would cause a machine check exception. Both (4) and (5) are grouped into a single result, where the downloaded code did not run, which from the attacker perspective is the same.

Detecting the error during code download (*Err-Protocol*) is possible as the protocol should error back all received data. In addition, once the correct number of bytes are received, the chip should stop echoing back data. Thus we can attempt to send a single additional byte to detect if the chip is still (erroneously) waiting on data. If we detect the chip still waiting on data, this suggests the variable holding the expected program length is corrupt.

5.3 Hardware Targets

In order to validate the attack, it is first performed on a development board, and then moved to a real-life ECU. An example of the two targets are shown in Figure 17.



(a) MPC5676R Development Kit

(b) ECU "Bench-Top" Target

Figure 17: Hardware targets used to validate the EMFI effectiveness.

5.3.1 MPC5676R Development Board

This target is a development kit available from the vendor. Here we used MPC57XXXMB for the base-board and XPC567XMB-PT for the variant specific target, shown in Figure 17a

The device configuration is as follows:

1. PLLCFG2 = 1 (board DIP switch).
2. External clock = 40 MHz crystal.
3. BOOTCFG1 = 0 (board DIP switch).

This will be reported as *5676DK* in the results tables.

5.3.2 ECU In-Situ

For the "in-situ" tests, the applicability of the exploit on the chip used on development boards is then compared to a real-life application. The real-life application chosen is a GM E41 ECU, part number 12691652, which is used in the Chevrolet Silverado 2500 HD & 2019 GMC Sierra 2500 HD. This device was chosen as it represents a recently released vehicle.

Compared to a simple laboratory setting, performing the attack on the "off-the-shelf" device requires providing power and communications to the target device, as well as reverse engineering of required test points.

1. PLLCFG2 = 1 (board configuration).
2. External clock = 40 MHz crystal.
3. BOOTCFG1 = 0 (via jumper wire soldered to board).
4. RESET pin connected via jumper wire inserted into board via.

This will be reported as *E41* in the results tables.

5.4 Results

Unless otherwise described, the following setup for the fault injection attack was used:

- EMFI voltage set to 444V.
- EMFI tool using 4mm, counter-clockwise (CCW) wound coil.
- Trigger based on time since end of the received echo of last character of the transmitted password.

An overall summary of the results on the four targets is given in Table 1. This shows the various responses achieved for sending the public password (shown in table as FEE.., full sent password was `FEEDFACECAFEBEEF`) and an invalid private password (shown in table as 112.., full sent password was `1122334455667788`. Note that when switching the changed sent password, *no physical changes to the device setup happen.* The only change is in the script performing the fault injection campaign.

Table 1: Overall success rate for target devices in 56xx family.

Result	5676DK 1122..	E41 4mmCW 1122..	E41 4mmCW FEED..	E41 FEED..
Normal	92.8%	98.5%	98.5%	91.5%
Err-Reset	0.10%	0.02%	0.04%	0.16%
Err-Protocol	0.00%	0.00%	0.00%	0.00%
Err-RunFail	5.85%	0.00%	0.00%	8.29%
Success	1.23%	1.26%	1.43%	0.00%

When changing between targets, the physical EMFI setup is moved to the next target. As our test setup allows real-time reviewing of results (such as *Err-Reset* vs *Normal* vs *Success*) we manually position the EMFI tip over various areas of the chip until successes are seen (where possible). For these devices the location is typically on or near the 'Freescale' logo on the chip markings.

The notation for the various targets was previously listed in each target summary section.

For the MPC5676R device, it is noted that sending the public or invalid password had generally similar success rates. For the E41 ECU, note the "4mm CCW" injection tip did not result in successful attacks, while that tip was successful on the two development boards. Of these results, the successful E41 attack was with the "4mm CW" injection tip. This may simply be a fluke related to positioning, but the attack was successfully repeated after re-positioning the board using the "4mm CW" injection tip, but not with the "4mm CCW" tip.

5.5 Securing ECU in presence of faults

In this example, the low-level bootloader of the microcontroller has not been hardened from a fault injection campaign. Thus the developer of the ECU is more limited in their defense mechanisms.

Additional testing suggested that when a device was configured with the "public" password, the fault injection attack was less successful. In this configuration the device allows SRAM access by default, but disallows flash memory access. Thus there is some danger here since this means *any* attacker will have access to the SRAM contents, which may be used for holding private keys.

A full description of the suggested countermeasures is given in the extended work.[16] This includes using the public password, along with scrambled internal flash passwords in case a single fault injection is used to enter the private password mode.

[16] Colin O'Flynn. "BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks". In: *Proceedings of ESCAR-EU*. Nov. 2020. URL: https://eprint.iacr.org/2020/937.pdf.

6 Conclusions

Electromagnetic fault injection is a useful tool when developing automotive electronic systems. As demonstrated in this paper, it can be used to inject faults into automotive systems for both safety & security testing.

For safety testing, EMFI is a powerful method of injecting faults into arbitrary platforms. This allows evaluation of targets which have not been instrumented for fault injection tests. EMFI is able to generate many of the faults in digital devices that are specified in Table 30 of ISO 26262-11:2018.

For security testing, EMFI allows bypassing of many common security mechanisms. As demonstrated here, it can be performed in typical "workbench" settings. This means that R&D engineers can easily access the tools, but also means they may be a practical threat as "advanced tuning garages" are likely to have access to such tools.

The ChipSHOUTER from NewAE Technology Inc. is a low-cost electromagnetic fault injection tool that can be used for these safety & security testing tasks. As the ChipSHOUTER works with the ChipWhisperer ecosystem, the advanced glitch mapping and triggering logic from tools such as ChipWhisperer-Pro can be used to insert faults at specific times in the execution of code within the device.

References

Anderson, Ross and Markus Kuhn. "Tamper resistance: a cautionary note". In: *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*. WOEC'96. USA: USENIX Association, Nov. 1996, p. 1. (Visited on 10/14/2020).

Avizienis, A and D.A. Rennels. "Fault-tolerance experiments with the JPL STAR computer." In: San Francisco, CA, Jan. 1972.

Avižienis, Algirdas. "Design of fault-tolerant computers". In: *Proceedings of the November 14-16, 1967, fall joint computer conference*. AFIPS '67 (Fall). New York, NY, USA: Association for Computing Machinery, Nov. 1967, pp. 733–743. ISBN: 978-1-4503-7896-3. DOI: 10.1145/1465611.1465708. URL: https://doi.org/10.1145/1465611.1465708 (visited on 10/14/2020).

Boneh, Dan, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults". en. In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by Walter Fumy. LNCS. Berlin, Heidelberg: Springer, 1997, pp. 37–51. ISBN: 978-3-540-69053-5. DOI: 10.1007/3-540-69053-0.

Gerlinsky, Chris. "Breaking Code Read Protection on the NXP LPC-family Microcontrollers". In: 2017. URL: https://recon.cx/2017/brussels/resources/slides/RECON-BRX-2017-Breaking_CRP_on_NXP_LPC_Microcontrollers_slides.pdf.

Gunneflo, U., J. Karlsson, and J. Torin. "Evaluation of error detection schemes using fault injection by heavy-ion radiation". In: *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*. June 1989, pp. 340–347. DOI: 10.1109/FTCS.1989.105590.

Koopman, Phil. *A Case Study of Toyota Unintended Acceleration and Software Safety*. en. 2014. URL: https://ptolemy.berkeley.edu/projects/chess/pubs/1081/koopman14_toyota_ua_slides.pdf.

O'Flynn, Colin. "BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks". In: *Proceedings of ESCAR-EU*. Nov. 2020. URL: https://eprint.iacr.org/2020/937.pdf.

— *Fault Injection using Crowbars on Embedded Systems*. Tech. rep. 810. 2016. URL: http://eprint.iacr.org/2016/810 (visited on 07/14/2020).

Sabo, Joseph D. and Joel A. Karp. "Radiation circumvention technique". Pat. US4413327A. Nov. 1983. URL: https://patents.google.com/patent/US4413327A/en (visited on 10/15/2020).

Zussa, Loic et al. "Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter". In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. May 2014, pp. 130–135. DOI: 10.1109/HST.2014.6855583.

DISCLAIMER