Injecting Software Vulnerabilities with Voltage Glitching

Yifan Lu
me@yifanlu.com

***Abstract*—We show how voltage glitching can cause timing violations in CMOS behavior. Then we attack a real, security hardened, consumer device to gain code execution and dump the secure boot ROM.**

I. Introduction

Glitching, or fault injection, has been used for over a decade [1] to attack software running on secure execution environments. Due to the upward trend in pricing in the software exploit market [8] and the increased hardening of security in consumer devices, there has been a rise in popularity of injecting faults to gain control of a device. Fault injections can be used to cause a malfunction in the target's system-on-chip (SoC) and, when the malfunction is controlled properly, can be used by an attacker to take full control of the device.

Voltage glitching is a specific kind of fault injection and is attractive because is it inexpensive to set up and is widely applicable to most chips [2]. *Crowbar voltage glitching* was introduced by O'Flynn [6] and implemented in the Chip-Whisperer open hardware platform to bring these attacks to the mainstream. It works by abusing the capacitance ringing effect caused by introducing a crowbar circuit into the existing system. The ringing causes faults that can be exploited.

A. Background

We looked at the prior attempts at modeling voltage fault injections and ordered them in terms of abstraction (see table I). Most recently, in a paper by Timmers, Spruyt, and Witteman [9], they created an architectural model of fault injection for ARM devices. Their model considers instruction corruption due to bit-flips caused by the fault. Their model is applicable to many kinds of fault injection.

Another paper we drew inspiration comes from Zussa, Dutertre, Clédière and Tria [10]. Their work focused on confirming empirically that the mechanism for faults induced by voltage glitching is due to setup/hold time violations. They concluded that voltage glitches increased the propagation time of combinational logic which creates setup/hold time violations. Another connection we make is that while their paper wanted empirical evidence for a wide-held belief in how voltage glitches work we want a rigorous theoretical model for the same wide-held belief.

One level down from that is the paper by Djellid-Ouar, Cathebras and Bancel [4] which concluded that D-flip-flops (and therefore most memory elements) were mostly immune to standard voltage attacks. Their paper analyzed bistable CMOS elements using the small signal model.

This work was not supported and do not represent the approval or rights of any third parties.

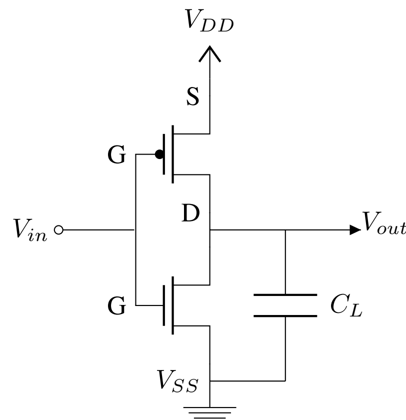

Fig. 1. Standard CMOS inverter with load capacitance C_L

In addition to modeling voltage glitches, there has been many experiments in applying voltage fault injections to affect processor operations on complex SoCs. The aforementioned paper from Timmers, Spruyt, and Witteman [9] details a bypass of a secure boot code integrity check on an ARM processor through voltage glitching. O'Flynn [6] used his crowbar method on a Raspberry Pi computer to modify the result of a running counter.

The PlayStation Vita was a hand-held gaming console released in 2012. It used a custom designed Samsung 45nm SoC [3]. The SoC includes a MeP architecture processor which we nicknamed "F00D," that performs cryptographic tasks and serves as the boot processor. The boot ROM used by F00D is unmapped early in the boot process and is then unable to be read out through pure software means.

Our contribution will be in two separate domains. In section II, we will analyze the CMOS transistor behavior in order to understand *when the combinational logic is most susceptible to voltage glitch induced faults*. Then, in section III we will apply our understanding to perform a fault injection attack on the PlayStation Vita's SoC to gain early (boot time) execution control of F00D in order to dump the boot ROM.

II. CMOS Voltage Glitch Model

To analyze the CMOS behavior during a voltage glitch, we will only consider the voltages near the gate itself. This

TABLE I
RELATED WORKS IN MODELING VOLTAGE GLITCHES

Paper	Year	Level	Applicability	Results
TSW [9]	2016	Architectural	ARM, any glitches	Faults modeled by corrupted instructions
ZDCT [10]	2013	Digital Logic	Voltage/clock glitching	Fault caused by setup/hold violations
DCB [4]	2006	Gates/Elements	Voltage glitching	D-flip-flops not susceptible to voltage glitches
This Paper	2018	Transistor	Voltage glitching	Effects are data dependent

TABLE II
NOTATIONS USED

V_{DD}	Supply voltage (normal operations)
V_{SS}	Ground voltage (typically 0 V)
V'_{DD}	Glitch supply voltage (typically ~ 0 V)
V_{in}	CMOS input voltage
V_{out}	CMOS output voltage
C_L	Gate load capacitance (simplified)
V_{SG}	Voltage from PMOS source to PMOS gate
V_{TH}	PMOS threshold voltage
V_{IL}	Switching threshold for input low
V_{IH}	Switching threshold for input high
R_{eqp}	PMOS equivalent resistance with source V_{DD}
R_{eqn}	NMOS equivalent resistance with source V_{SS}
R'_{eqp}	PMOS equivalent resistance with source V'_{DD}
t_{pHL}	Propagation delay for output going low
t_{pLH}	Propagation delay for output going high
τ_A	Glitch start time
τ_B	Glitch end time
t_G	$\tau_B - \tau_A$
t_{gHL}	Propagation delay for output going low during glitch (to be defined)
t_{gLH}	Propagation delay for output going high during glitch (to be defined)

simplification will disregard everything that happens when the voltage pads on the IC is suddenly changed. For example, if a crowbar circuit is used to quickly short V_{DD} to V_{SS} for some amount of time, we do not actually observe a short at the MOSFET. Instead, the capacitance and the power-delivery-network of the circuit will create a ringing effect [6] that will be observed at the MOSFET. Our analysis will therefore only consider the duration and amplitude of these rings as the "glitch" and not the source of them. We note that a more in depth analysis can incorporate such external effects without affecting our understanding of what happens at the MOSFET.

A. Notation

We will use standard notation where applicable. For convenience, they are defined in table II along with other labels relevant in our analysis.

B. Motivation

Consider a standard 1-input CMOS gate (an inverter, figure 1). Let V_{in} be the input voltage and V_{out} be the output. We define a voltage glitch to be a span of time from τ_A to τ_B during which, we set $V_{DD} \leftarrow V'_{DD}$. V'_{DD} is the glitch voltage (and ideally $V'_{DD} = V_{SS}$). $t_G = \tau_B - \tau_A$ is the glitch width. C_L represents the load capacitance and includes the gate parasitic capacitances, the wire capacitance, and the input capacitance of the next gate.

We will focus on the span of time between τ_A and τ_B when the voltage glitch happens. Looking at only one inverter, we

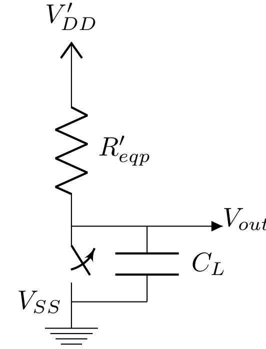

Fig. 2. Equivalent circuit for PMOS in non-saturation.

see that in that span of time, the input could either toggle at least one time or not toggle at all. We will analyze the behavior of the inverter in both cases to see when the output value can be influenced by the glitch.

C. Non-toggling

First, consider the case when V_{in} does not change during the voltage glitch. If the input is a logical '1' ($V_{in} \geq V_{IH}$) then the output would be at $V_{out} = V_{SS}$. Since $V_{SG} < V_{TH}$, the PMOS is off and there is no source for V_{out} to change. Therefore the voltage glitch caused no change in behavior.

If the input is at logical '0' ($V_{in} \leq V_{IL}$), then before the voltage glitch, $V_{in} = V_{SS}$ and $V_{out} = V_{DD}$. When the glitch happens, the PMOS will be on, so we can use the equivalent resistance model defined in Chapter 5.4 of [7] to analyze the dynamic behavior.

Using equation 5.17 in [7], we find the fall time to be

$$t_{gHL} = \ln(2) R_{eqp} C_L \tag{1}$$

where R_{eqp} is the on-resistance of the PMOS at the glitch voltage V'_{DD}. Note that this is slightly different from the commonly used value t_{pHL} (fall time of the inverter) because t_{pHL} is defined with respect to R_{eqn}, the on-resistance of the NMOS.

The output goes to '0' for the duration of the glitch and then is be restored to '1' as V_{DD} is restored to its pre-glitch value. This means that if the input is '0' at the start of the voltage glitch and does not toggle, then the output will be low from $\tau_A + t_{gHL}$ to $\tau_B + t_{pLH}$ (assuming $t_{gHL} \ll t_G$). Note this is similar to a static hazard, which can cause setup and hold time issues.

D. Toggling

Consider a transition from a logical ‘0’ to ‘1’ at the input during the voltage glitch (from τ_A to τ_B). Since this turns off the PMOS, there should be no change in behavior at the output (compared to what happens if it toggles while there is no voltage glitch). However, if the transition is from a logical ‘1’ to ‘0’, then according to [10], the rise time of the inverter output, t_{gLH} increases as V'_{DD} decreases. Additionally, if the glitch voltage $V'_{DD} < V_{TH}$, then the output will not go high for the duration of the voltage glitch (t_{gLH} would be infinite). So this means the propagation delay increases as V'_{DD} decreases up to t_G. This also causes setup and hold time issues.

E. Results

In both cases, we see that the delay introduced to a single gate by a voltage glitch is composed of a rise/fall time (t_{gHL} or t_{gLH} depending on the input value), the glitch width t_G, and finally the delay for when V_{DD} is “restored” and the “correct” input must propagate to the output again (t_{pLH}). We can define t_g to be the propagation delay of that inverter during a voltage glitch from the input to the output (we consider the value to be “propagated” only when the output has the correct value). From above, we see this can be broken into four cases.

$$\begin{aligned} t_{g,\text{non-toggle},0} &\geq \begin{cases} t_G + t_{pLH}, & \text{if } t_{gHL} > t_G \\ t_G - t_{gHL} + t_{pLH}, & \text{otherwise} \end{cases} \\ t_{g,\text{non-toggle},1} &= 0 \\ t_{g,\text{toggle},0\text{-to-}1} &\geq \begin{cases} t_G + t_{pLH}, & \text{if } t_{gLH} > t_G \\ t_{gLH}, & \text{otherwise} \end{cases} \\ t_{g,\text{toggle},1\text{-to-}0} &= t_{pHL} \end{aligned} \tag{2}$$

Now, if we have a chain of N inverters, we can find the total propogation through the chain considering that only the first inverter is affected by the voltage glitch (with delay $t_{g0} = (t_{g,\text{non-toggle},0} + t_{g,\text{toggle},0\text{-to-}1})/2$, the average delay of the two cases that are affected by a voltage glitch). Note that if two inverters in a chain are affected, we do not care about the status of the later inverter because the earlier one will already propagate its corrupted value and later corrected value down the chain.

$$t_g(N) = t_{g0} + (N-1)\frac{t_{pLH} + t_{pHL}}{2} \tag{3}$$

In practice t_{gHL}, t_{gLH}, t_{pHL}, t_{pLH} will all be very small compared to t_G[1]. Therefore we can simplify equation 3 to:

$$t_g(N) \geq t_G + N\frac{t_{pLH} + t_{pHL}}{2} \tag{4}$$

Note that the second part is just the CMOS propagation time with fanout 1. That means as long as t_G is much greater than the rise/fall time of the output, the propagation delay is bounded by the CMOS delay plus the glitch width.

Of course in reality, the analysis gets a lot more complicated. First, the voltage glitch will not affect every CMOS at the same time. There will be a sort of “propagation delay” of the voltage change itself. Second, when we consider 2-input gates and higher, there could be a mixture of non-toggling and toggling behavior at each gate. Then of course, there are the non-linear capacitance that makes computing C_L difficult.

However, there are some conclusions we can draw from this analysis.

- Asynchronous circuits are most affected by voltage glitches due to the introduction of hazards.
- Synchronous circuits are not immune if the voltage glitch cause a setup/hold time violation.
- Critical paths can be extended if a voltage glitch happens at the right time (0-static or 1-toggling).
- Long critical paths are the best targets for voltage glitching (i.e: processor ALU).

III. VITA GLITCHING

The firmware and boot loader are found on an external eMMC storage, which has logical sectors 512 bytes wide. Upon boot, sector 0, the master boot record is read. The Vita’s MBR is a custom format not used in any other device. The details of this MBR format are beyond the scope of this paper, but two fields are of importance. `bldr_offset` is at MBR offset `0x30` and `bldr_size` is at MBR offset `0x34`. These two fields are both 4 bytes wide and both the offset and size are defined in number of sectors. They are used by the boot ROM to determine where the boot loader is located.

One thing we discovered early on (through trial and error) was that if `bldr_size` > `0xDE`, then an assertion fails and the device is rebooted. Otherwise, the eMMC is read starting at the offset for `bldr_size` blocks. Our hypothesis is that there is a fixed size buffer that the boot loader is read into which necessitates the size check. If we use a fault injection to bypass this check, we can introduce a buffer overflow vulnerability.

A. Experimental Setup

There were three main components to our setup. First, we needed a way to monitor the eMMC traffic and use it as a trigger for the voltage glitch. Second, we needed a way to perform the voltage glitch. Finally, we wished to automate the steps in order to find the optimal parameters for glitching.

Fortunately, the ChipWhisperer gave us an easy way to do all of this. The hardware has a MOSFET that performs the crowbar voltage glitch [6]. The open hardware design allowed us to implement a custom eMMC trigger for the MOSFET (see appendix A-A). Finally, because the timing and duration of the voltage glitch is highly dependent on the power distribution network of the device [6], it is difficult to compute the optimal timing parameters for a successful glitch on the size check. Instead, we exhaustively searched for the timing offset and width of the crowbar activation after being triggered that results in a successful fault injection.

[1] t_{pHL} and t_{pLH} are around the order of 30 ps for 0.25 μm CMOS technology [7].

Additionally, we made sure to synchronize the ChipWhisperer's glitch module clock with the device's external clock input. This way we can make sure the two devices are in phase and decrease the variance in finding working parameters.

B. Parameters

For a successful fault injection, we need to find parameters f, the clock frequency of the glitch module and the Vita's external clock input, N, the number of cycles after seeing the eMMC trigger before activating the crowbar circuit and M, the number of cycles to hold the crowbar on before releasing it. Note that these parameters will determine the response of the ringing effect that will ultimately cause a fault in the size comparison[2].

We know from experimentation[3] that F00D boot ROM runs at $f_{clk} = \frac{1}{9}f$ (where f is the external clock input). A faster clock will increase the chance of a successful fault (due to timing violations). In practice, we cannot over-clock much past the default rate of 37MHz or we will run into non-voltage glitch related timing violations that prevent the circuit from working properly[4]. However, in our case, because of the noisy design of our glitching setup, we picked $f = 12\,\text{MHz}$ since it was the fastest we were able to go without running into a variety of signal integrity issues.

For N and M, we chose to brute force every possible value to find ones that worked.

For the parameter search, we first used manual analysis to get a ballpark idea of when `bldr_size` is being checked. We hypothesized that the check must happen after the response packet for the eMMC `READ_SINGLE_BLOCK` request for the MBR block. First we narrowed the window for the search to be the period of time between commands. We measured the amount of time from the response of `READ_SINGLE_BLOCK` packet (with a valid `bldr_size`) to the request of the `SEND_STATUS`[5] packet using a Rigol DS1054Z oscilloscope. At $f_{clk} = 12/9\,\text{MHz}$, it took $36\,\text{ms}$[6]. This yields an upper bound for N.

When an attempt fails with a particular N, M pair, we observe one of the following behaviors:

- The device halts. Usually we only see this with very large M so it's likely the CMOS is losing power and shutting off.

[2]Many sources mention removing decoupling capacitors for better result without giving a detailed reason. We were able to get voltage glitches to work both with and without removing the decoupling capacitors. It is our belief that removing the decoupling capacitors changes the response of the ringing and therefore the parameters for a successful glitch. But in our case, it does not make it any more or less tractable.

[3]Toggle GPIO and measure with the Rigol DS1054Z.

[4]This might be prevented with, for example, better cooling but we did not go down this avenue.

[5]Defined as part of the eMMC read procedure [5].

[6]We discovered that due to a bug, the boot ROM spins the processor to wait for the `READ_SINGLE_BLOCK` request to complete. However, the smallest granularity of the spin time is about 1000 times the average amount of time it takes for the command to complete (as we observed on the DS1054Z). So to make things easier, our brute force actually ran backwards starting from the far end of the window.

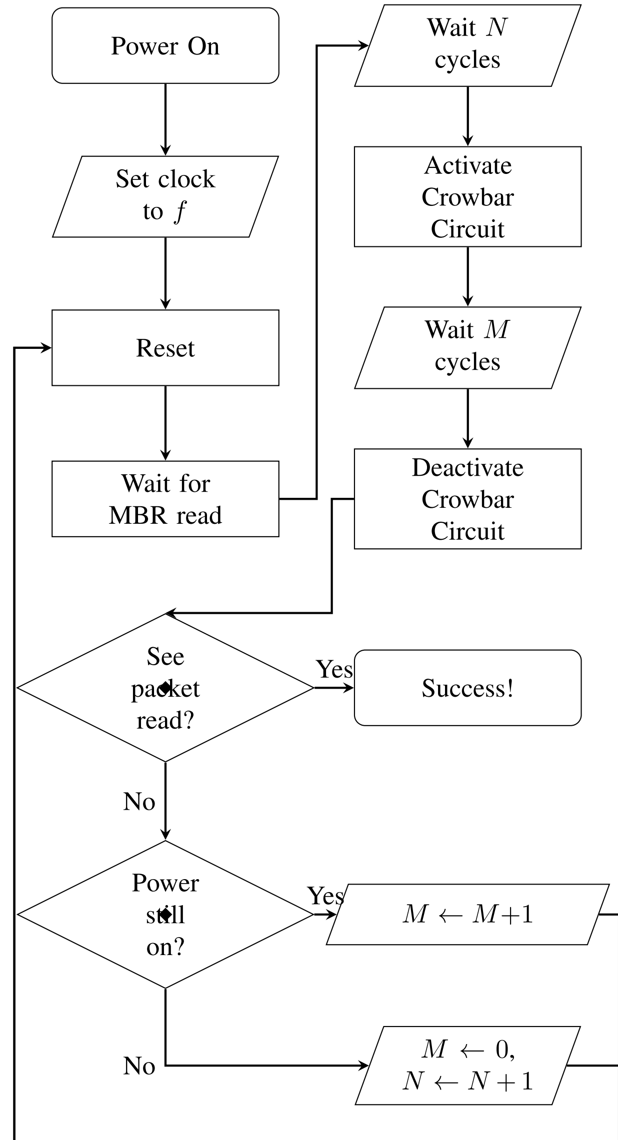

Fig. 3. Parameter search process

- The device reboots and we observe the eMMC `GO_IDLE_STATE` packet. This is the most common observation[7].
- The device goes into an unknown state and makes an unexpected request (and possibly restarts).
- The device requests the first block of the bootloader (this is the success case).

We therefore only need to record the next packet seen after reading MBR (or time out) and check if it is the first block of the boot loader to indicate success. Figure 3 summarizes the

[7]We believe this is due to the fault happening at the wrong place or not happening at all. Both of which would cause an assertion to fail. Originally we wanted to try some timing attack to differentiate the two cases. However we later found out that the designers actually anticipated this and masked any reboot triggered by an assertion fail to first spin for a random number of cycles determined by a TRNG to make it hard to determine "which" assertion fail caused the reboot.

TABLE III
RANGE OF WORKING P

Parameter	Units	Min	Max
f	Hz	12	12
N	cycles	40800	40820
M	cycles	45	55

process.

C. Results

After writing a script with the ChipWhisperer API to try all possible N and M values (see appendix A-B) and running it overnight, we found a successful case with the following parameters (see table III). Due to the effects of wire capacitance and external capacitance, the parameters are highly specific to our setup and environment. However, after finding a valid N,M pair, even with environmental variations, we can find another valid N,M pair close by. If the equipment and target board and not moved or touched at all, we can reproduce the injected fault with the same N,M for $> 80\%$ of the time.

D. Exploiting

We managed to find the glitch parameters N and M that faults the `bldr_size` check. However this only gives us a vulnerability. We still need to exploit it. Fortunately, this was made easy when we observed that when the glitch was successful, we see exactly `0xE2` blocks read if `bldr_size` $\geq$ `0xE2`. If `bldr_size` $<$ `0xE2`, then it will read `bldr_size` blocks exactly (with a successful glitch). Therefore we guessed that we were overwriting the currently executed code and that guess was correct. With that, we launched a payload that dumped everything we can read through UART.

Unfortunately we then discovered the boot ROM does not seem to be mapped anywhere. It appears that hardware copies the contents of boot ROM to SRAM and the reset vector points directly to SRAM. That means the boot ROM is able to clean up parts of itself as it executes. To get the remaining parts (that were cleaned up), we had to glitch different parts of the boot ROM (running in SRAM) and gain code execution earlier and earlier on. Each time we dump more code, we gain more information and can develop more specific glitch targets. The details of these additional injected faults and their subsequent exploitation is beyond the scope of this paper.

IV. CONCLUSION

By looking at how voltage glitches introduce timing violations into a digital circuit, we can find good snippets of code to glitch. Once a target is found, we can search for the right timing parameters for our crowbar circuit to cause a fault. We do an exhaustive search because it is difficult to predict how changing the parameters N and M actually affects the CMOS circuits. Finally, the injected fault introduces a software vulnerability that can be exploited to gain code execution. All of this can be done at a low cost thanks to the open hardware interface of the ChipWhisperer. With a custom script written for ChipWhisperer, we created a working attack on a security hardened consumer device.

REFERENCES

[1] Ross Anderson and Markus Kuhn. “Low cost attacks on tamper resistant devices”. In: *Security Protocols*. Ed. by Bruce Christianson et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 125–136. ISBN: 978-3-540-69688-9.

[2] A. Barenghi et al. “Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures”. In: *Proceedings of the IEEE* 100.11 (Nov. 2012), pp. 3056–3076. ISSN: 0018-9219. DOI: 10.1109/JPROC.2012.2188769.

[3] Andrew Cunningham. *Samsung to Manufacture PS Vita SoC*. July 29, 2011. URL: https://www.anandtech.com/show/4553/samsung-to-manufacture-ps-vita-cpu (visited on 12/19/2018).

[4] A. Djellid-Ouar, G. Cathebras, and F. Bancel. “Supply voltage glitches effects on CMOS circuits”. In: *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.* Sept. 2006, pp. 257–261. DOI: 10.1109/DTIS.2006.1708651.

[5] *Embedded MultiMediaCard(eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, Sleep Modes, Dual Data Rate, Multiple Partitions Supports and Security Enhancement (MMCA, 4.4)*. JESD84-A44. JEDEC Solid State Technology Association. Mar. 2009.

[6] Colin O’Flynn. “Fault Injection using Crowbars on Embedded Systems”. In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 810.

[7] Jan Rabaey and Anantha Chandrakasan. *Digital Integrated Circuits: A Design Perspective*. Pearson Education, 2003.

[8] Marc Ruef. *Exploit Pricing: Analysis of the Market in Digital Weapons*. Oct. 13, 2016. URL: https://www.scip.ch/en/?labs.20161013 (visited on 12/18/2018).

[9] N. Timmers, A. Spruyt, and M. Witteman. “Controlling PC on ARM Using Fault Injection”. In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Aug. 2016, pp. 25–35. DOI: 10.1109/FDTC.2016.18.

[10] Loïc Zussa et al. “Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism”. In: July 2013, pp. 110–115. DOI: 10.1109/IOLTS.2013.6604060.

APPENDIX A
CODE

A. Custom ChipWhisperer with eMMC Triggering

https://github.com/TeamMolecule/chipwhisperer

B. Parameters Search Script

https://github.com/TeamMolecule/petite-mort