SRAM Has No Chill

Attack:

SRAM has traditionally been considered resilient against cold boot attacks, as it typically loses its stored information once power is removed. However, in this paper, the authors introduce a novel approach that challenges this assumption, demonstrating that on-chip SRAM can, under specific conditions, retain data even after power cycles and software resets. This new method, called Volt Boot, exploits a fundamental vulnerability in modern system-on-chip (SoC) architectures, specifically targeting the way power distribution networks are physically separated within these chips. Unlike conventional cold boot attacks that rely on extreme cooling or the natural retention time of volatile memory, Volt Boot manipulates power states—such as selectively keeping certain sections of the chip powered while others are turned off—to achieve data persistence across reboots.

By conducting experiments on multiple ARM Cortex-A devices, the researchers successfully applied Volt Boot to extract data stored in caches, processor registers, and internal RAM (iRAM) with complete accuracy. This is a significant departure from prior SRAM-based attacks, which often require intricate signal processing or statistical reconstruction methods to retrieve meaningful information. In contrast, Volt Boot is capable of directly recovering stored data with 100% fidelity, making it a powerful and highly reliable threat. The implications of this discovery are substantial, as they expose a previously unrecognized risk in the security of modern processors, particularly in environments where sensitive data is assumed to be erased upon reboot.

2.1 On-Chip SRAM

SRAM is the building block of volatile internal memories, such as caches, iRAM, registers, TLBs, and BTBs, making them one the most common memory in modern computing systems. Figure 1 illustrates a typical 6-transistor SRAM cell, which is composed of two inverters in a positive feedback configuration to hold a data bit. The cell's state is accessible through transistors N1 and N2, and these transistors provide access to the data bit (Q) and its complement (~Q), respectively. Unless a processor executes a read/write command, *Word Line* remains de-asserted with data stored in the cross-coupled structure formed by inverter ① and inverter ②.

SRAM is energy-efficient, fast, long-life, and self-refreshing; the only requirement for data retention is sufficient voltage from the power supply to maintain the positive feedback loop between the two inverters. The voltage required by an SRAM cell to retain the state is called its data retention voltage [20]. Data retention voltage is both process variation and data-dependent but is generally much lower than the threshold voltage of either inverter (i.e., the 'turn-on' voltage). Provided the voltage of a cell is more than or equal to its data retention voltage, the cell retains its state. Exploiting this property, modern processors dynamically scale down the voltage when the RAM is not actively accessed because it reduces the *energy leakage* through parasitic paths.

Unlike other types of memory, *direct access* (*i.e.,* direct software read/write) to many types of on-chip SRAM (e.g., instruction cache) is uncommon. However, most SoCs provide access to these internal memories through various methods to debug low-level memory errors; ARM allows RAMINDEX [23] and direct memory access [4] operations in ARMv8 architecture, while RISC V processors [38] allow memory-mapped access. For example, a Cortex-A72 processor provides access to 15 different internal RAMs, including caches, TLBs, and BTBs through its *cp15* co-processor interface.

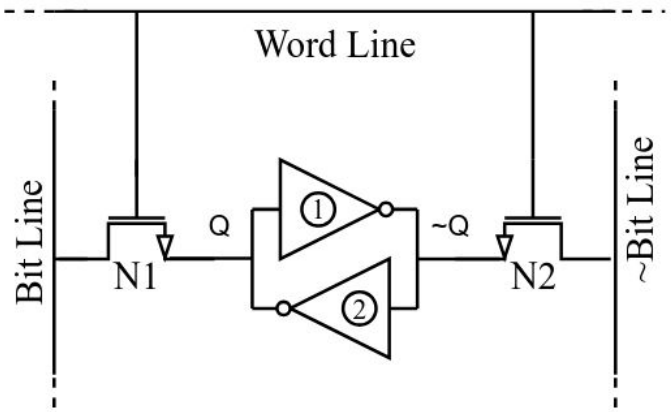
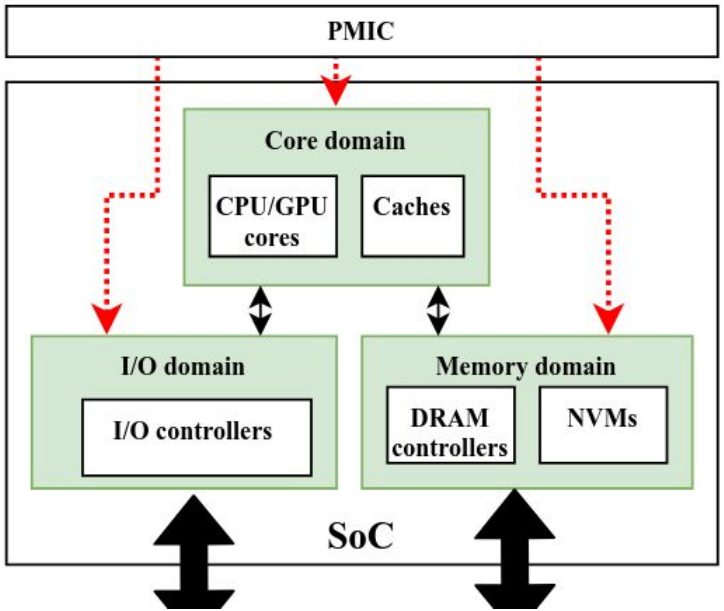


Figure 1: Typical 6-transistor SRAM cell.

Power Domains



Volt Boot System

6.1 Attack Execution Steps

In this section, we discuss how to execute an attack on SoCs (see Figure 5 for summary).

(1) **Identifying target domains and their associated pins:** Once we identify a target device, the first step is to identify the pins that supply SRAM with power. In most cases, it is impossible to locate a specific pin on the circuit board, because SoC chips use advanced packaging, such as BGA. However, it is not essential to find exact pins in an SoC package as supply pins are connected to passive components (*e.g.,* decoupling capacitors) or circuit-board-level test pads, which tend to be located near the PMIC (§5). The layout of passive components follows a typical pattern illustrated in Figure 4. For our evaluation platforms, we list the test points and target domain's pin names in Table 3 and present them in a visual form in Figure 6.

(2) **Attaching a voltage probe:** We measure the nominal voltage at the pin(s) and attach an external power supply probe at the same voltage level. The power source needs to supply sufficient current so that the level stays the same when we turn off the device's main power; otherwise, we risk losing the data. As an example, a Raspberry Pi 4 draws current through the test pad TP15 when we attach an 800mV probe. The current varies between 400mA to 600mA depending on software workload. When the SoC's main supply line (powered through a USB C) is disconnected abruptly, the cores draw power from the attached probe. The probe maintains the voltage level even if the cores demand a momentary current surge. The current consumption drops to 8mA after a few microseconds, and the memory domain stays in this retention state indefinitely.

(3) **Power cycling and booting the system:** Once the external probe is in place, we disconnect the device from the main power source while our voltage probe keeps the target SRAM active.

A system's boot-up method after power disconnect varies. Some systems allow booting up from alternative media only if the user data from the disk is erased, whereas some devices boot internally without needing any external boot media. We emulate this behavior by booting up the Raspberry Pis from another media through a USB mass storage device. We write a post-reboot data extraction program that performs the following tasks:

(A) Reduce contamination on the SRAM's retained data during boot-up by avoiding storing data to it (either explicitly or implicitly).

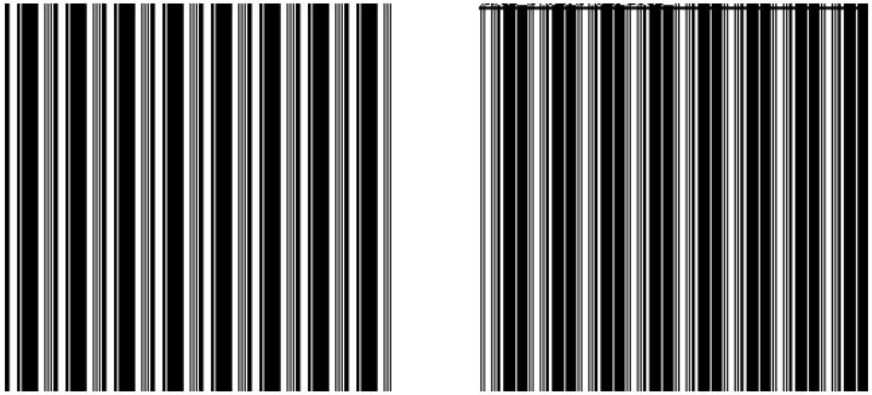
(B) Exfiltrate data from the SRAM to other memory (e.g., Flash, DRAM, or a debugger) for post-processing.

The cache extraction software executes CP15 instructions and reads out the *data register interface* of the caches to general-purpose CPU registers. Cache access requires read-/write to system registers. For processors with out-of-order execution capability, we must use appropriate data and instruction synchronization barriers. For example, Cortex-A72 processors uses `SYS #0, c15, c4, #0, <xt>` instruction to execute RAMINDEX operation (cache access request to CP15 co-processor). Data and instruction synchronization barrier instructions `DSB SY` and `ISB`, respectively, must follow this instruction before reading the cache data output register interface. A set of general load/store instructions moves the data from the general-purpose CPU registers to DRAM for further processing.

We directly dump the iRAM's through the debug interface, because i.MX535 requires no external firmware support for booting up. Thus, we connect a JTAG probe and directly read out the processor's (Cortex-A8) iRAM contents.

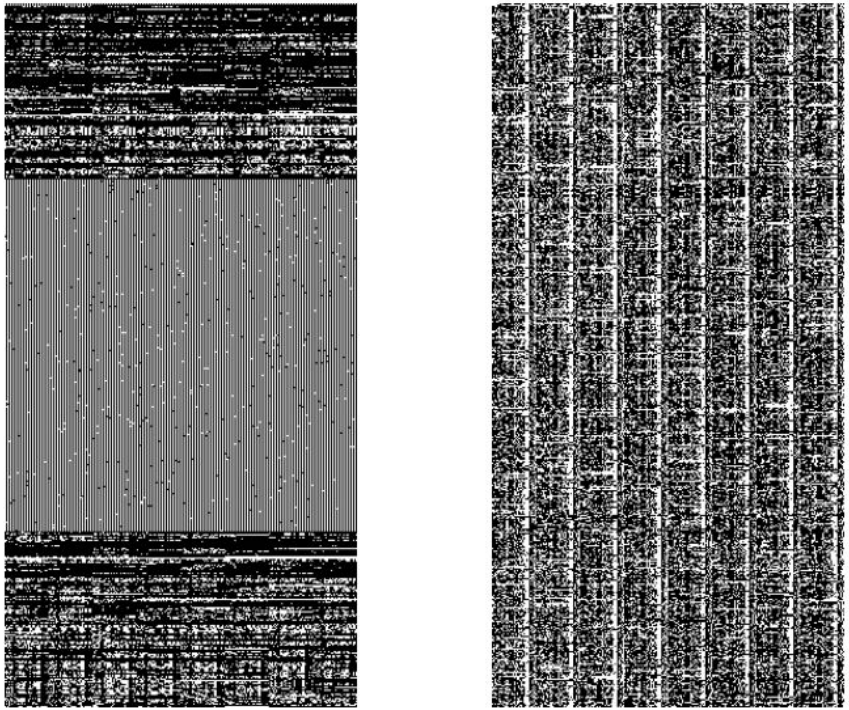
(4) **Analysing the memory contents:** Depending on the target SRAM and the objective, an attacker needs to adapt post-processing. Since *Volt Boot* reads out the memory without any error, the noise source in a successful attack is the *dynamic behavior* of software and its effects on the data stored in embedded SRAM. For example, error-free key extraction from a cache memory depends on the processing core's workload and other background processes.

Results



(a) (b)

Figure 7: Snapshots of i-cache after attacking bare-metal software in (a) BCM2711 and (b) BCM2837 SoCs. Uninitialized cache cells power on into random sate (see Figure 3), but when we execute *Volt Boot* attack, instructions stay in the i-cache across power cycles.



(a) d-cache (b) i-cache

Figure 8: Snapshots of the caches after executing *Volt Boot* on a system running a general application. We generate the cache images from one WAY of each type of cache.

Countermeasures

Potential Countermeasures to Volt Boot Attack

1. Eliminating Power Domain Separation (Not Practical)

- Power domain separation is crucial for power efficiency and performance.
- It allows dynamic voltage scaling and power gating in an SoC.
- Exposed pins in power domains help filter noise and stabilize voltage.
- **Downside:** Removing power domains is impractical due to performance and implementation constraints.

2. Purging Residual Memory (Not Effective)

- A simple method to prevent data retention is **erasing memory** during power-down.
- Software/hardware-driven memory clearing can be implemented.
- **Limitation:** Abrupt power disconnection stops all operations, preventing proper memory purging.

3. Resetting SRAMs at Startup (Effective but Rare)

- SRAM retention becomes useless if memory is cleared after reboot.
- Hardware-driven methods like **MBIST (Memory Built-In Self-Test)** can reset memory.
- **Observation:** Most devices boot with an undefined SRAM state unless software writes to memory.
- ARMv8-A **TrustZone (TZ) support** prevents unauthorized memory access.
    -

4. Mandated Authenticated Boot (Effective but Complex)

- Volt Boot attack requires booting a device with an exploitable system image.
- Secure boot mechanisms:
    - **OEM-signed system images** prevent unauthorized boot media.
    - Hash of the image is burned into fuses for verification.
- **Limitations:**
    - Not all devices enforce **authenticated boot**, as it complicates firmware updates.
    - Some processors assume **on-chip SRAM does not retain data** across power cycles.

5. Secure Cache Resetting (Challenging for L1 Caches)

- Some processors allow **L2 cache reset** by toggling nL2RST pin low for 16 cycles.
- **L1 caches cannot be reset** in this way due to their dependence on tag RAMs.
- Zeroization via ISA is a software solution but not hardware-based.
- Alternative: **Internal power toggling of SRAMs at reset** (requires hardware modification).

6. TrustZone Support (Effective for Memory Isolation)

- **ARM TrustZone (TZ)** is a hardware-backed security mechanism in Cortex-A processors.
- Every memory access undergoes a **hardware security check**.
- Cache memory lines are assigned a **security bit (NS)** to restrict unauthorized access.
- When TZ is enforced:
    - Secure memory remains **inaccessible across power cycles**.
    - Unauthorized access attempts trigger **hardware exceptions**

Critiques

Positives: Shows a novel method of using cold boot style attacks to retrieve data that is cached and does not require low temperatures.

Negatives: the paper is very technically dense

Slide Improvements

Add background

Shorted slides

Do not include direct text from paper

Add explanation to images

Do not have too much information on a single slide

Have a strong narrative through slides